

# **DESCRIÇÃO**

Tecnologias para particionamento e formatação de sistemas de arquivos e sua implementação e gerenciamento no sistema operacional Linux.

## **PROpósito**

Compreender a importância da implementação e o gerenciamento de sistemas de arquivos pelos principais sistemas operacionais do mercado, objetivando determinar o melhor sistema para cada situação, assim como realizar sua gerência de forma eficaz.

## **PREPARAÇÃO**

Antes de iniciar o estudo do tema, é desejável ter acesso a um computador (ou máquina virtual) com Linux instalado. Para os exemplos, foi utilizado o Ubuntu Desktop 20.04 LTS.

## **OBJETIVOS**

### **MÓDULO 1**

Identificar como são implementados os sistemas de arquivos

### **MÓDULO 2**

Aplicar os conceitos de sistemas de arquivos

## MÓDULO 3

Empregar as ferramentas de gerenciamentos de arquivos do Linux

## MÓDULO 4

Descrever o funcionamento dos principais editores de arquivos do Linux

# INTRODUÇÃO

Os sistemas operacionais são softwares essenciais para quaisquer dispositivos computacionais. Eles permitem que haja maior facilidade de uso por parte dos usuários, disponibilizando interfaces amigáveis assim como recursos para que as tarefas dos desenvolvedores sejam menos árduas.

Uma das tarefas que são desempenhadas pelos sistemas operacionais é fornecer um sistema que garanta a persistência das informações tratadas, armazenando-as nos dispositivos intermediários, como os discos rígidos magnéticos. Essas tarefas são implementadas pelos sistemas de arquivos.

Para compreender tal funcionalidade, estudaremos os conceitos de sistemas de arquivos e as diferentes formas de implementação, com ênfase nas vantagens e desvantagens de cada uma. Para isso, veremos as técnicas de formatação que foram desenvolvidas e conheceremos como se dá a implementação do sistema de arquivos do Linux.

De forma a solidificar esse conhecimento, aprenderemos sobre discos e partições, e como estas últimas são criadas em um sistema Linux. Veremos ainda como fazer a formação e a montagem dos sistemas de arquivos.

Uma vez criado tal sistema, vamos estudar as principais ferramentas que um administrador deve conhecer para manter o funcionamento. Por fim, conheceremos os principais editores de arquivos que podem ser utilizados para a configuração e a manutenção de um sistema Linux.

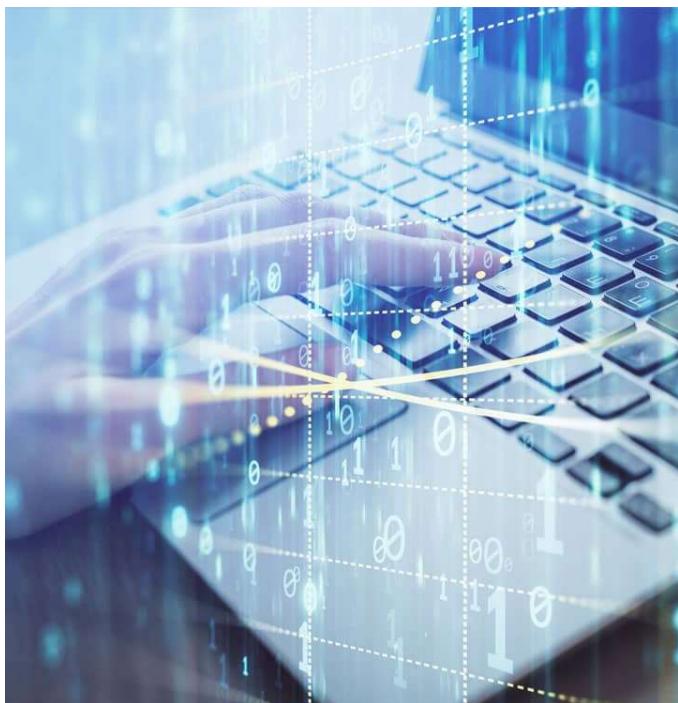
## MÓDULO 1

- 
- Identificar como são implementados os sistemas de arquivos

# CONCEITOS

Quando estamos utilizando um sistema computacional, é comum que, ao encerrar nossas atividades, guardemos o trabalho realizado para continuarmos em um momento posterior. Portanto, o armazenamento e a recuperação de informações são atividades essenciais para aplicações.

As principais exigências para armazenamento de informações são:



Deve ser possível armazenar uma grande quantidade de informações.



A informação deve sobreviver à finalização do processo que a utiliza (deve ser persistente).



Múltiplos processos devem ser capazes de acessar as informações concorrentemente.

É mediante a implementação de arquivos em discos ou outras mídias que o sistema operacional estrutura e organiza essas informações.

## 💡 VOCÊ SABIA

A parte responsável por essa gerência é denominada **sistema de arquivos**.

A manipulação de arquivos deve ocorrer de maneira uniforme, independente dos diferentes dispositivos de armazenamento.

De uma forma simplista, bastaria pensar em um dispositivo de armazenamento como uma sequência linear de blocos de tamanho fixo que dão suporte a duas operações:

Ler o bloco k;

Escrever no bloco k.

No entanto, essas são operações inconvenientes quando utilizadas em sistemas com muitas aplicações e possivelmente múltiplos usuários.

Algumas questões que surgem são:

Como saber em qual bloco se encontram as informações que necessitamos?

Como impedir que um usuário leia os dados de outro usuário?

Como saber quais blocos estão livres?

O sistema de arquivos é constituído de duas partes distintas:

Conjunto de arquivos

Armazena dados.



Estrutura de diretórios

Organiza e fornece informações sobre os arquivos do sistema.

Arquivos são unidades lógicas de informação criadas por **processos**. Um disco normalmente conterá milhares ou mesmo milhões deles, cada um independente dos outros.

## PROCESSOS

É um programa em execução em um sistema operacional, incluindo seu contexto de hardware (estado do hardware em determinado momento), valores de variáveis e seu espaço de endereçamento (região de memória que o processo pode utilizar).

## ARQUIVOS

Computadores podem armazenar dados em diferentes dispositivos de armazenamento, porém o usuário precisa acessar os dados contidos nestes dispositivos da mesma forma, independente do seu tipo.

### RESUMINDO

Para que um sistema possa ser usado de forma conveniente, o sistema operacional deve oferecer uma visão lógica e uniforme do dispositivo de armazenamento.

Um arquivo é constituído de informações, podendo representar programas ou dados. Um programa contém instruções compreendidas pela UCP (arquivo executável), enquanto um arquivo de dados pode ser estruturado livremente (podem ser numéricos, alfabéticos, alfanuméricos ou binários).

Um arquivo é identificado por intermédio de um **nome**, e as regras para os nomes variam de sistema para sistema.

## SAIBA MAIS

Em alguns sistemas operacionais, o nome do arquivo é composto por duas partes separadas com um ponto. A parte após o ponto é denominada **extensão do arquivo** e tem como finalidade identificar seu conteúdo.

As principais diferenças entre as regras para os nomes de arquivo são:

Quantidade máxima de caracteres;

Diferenciação entre caracteres maiúsculos e minúsculos;

Uso de caracteres especiais;

Nomes com extensão tendo significado ou não.

Em alguns sistemas, as extensões de arquivos são apenas convenções e não são impostas pelo sistema operacional.

## EXEMPLO

Um arquivo chamado “file.txt” pode ser algum tipo de arquivo de texto, mas aquele nome tem mais função de lembrar o proprietário do que transmitir qualquer informação real para o computador. Por outro lado, um compilador de Linguagem C pode exigir que os arquivos que ele tem de compilar terminem em “.c”. O sistema operacional, no entanto, não se importa.

Ao criar um arquivo, o processo deve atribuir um nome a ele para que, quando termine sua execução, o arquivo continue existindo e possa ser acessado por outros processos pelo mesmo nome. Ao receber um nome, o arquivo se torna independente do processo, do usuário e até mesmo do sistema que o criou.

# ESTRUTURA DE ARQUIVOS

No momento da criação de um arquivo, é possível definir qual organização será adotada. Esta organização pode ser uma estrutura suportada pelo sistema operacional ou definida pela própria aplicação.

## SEQUÊNCIA DESESTRUTURADA DE BYTES

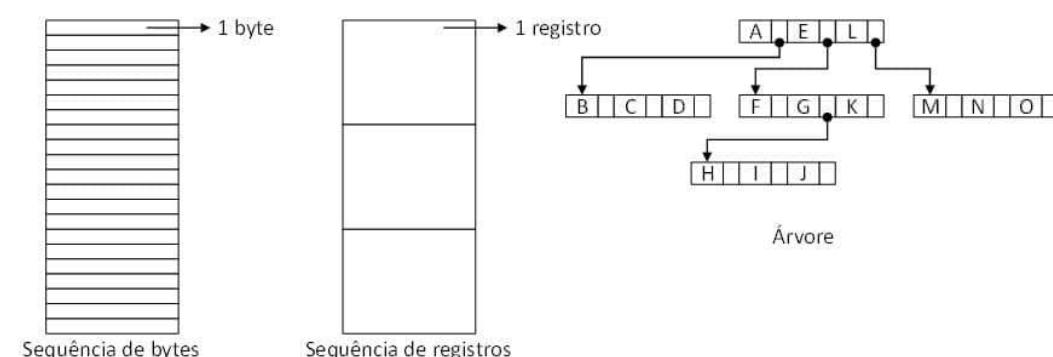
É a forma mais simples de organização de arquivos. Nesse tipo de organização, o sistema de arquivos não impõe nenhuma estrutura lógica para os dados (a aplicação deve definir toda a organização). A grande vantagem é a flexibilidade para criar diferentes estruturas de dados, mas todo o controle de acesso ao arquivo é de inteira responsabilidade da aplicação. Qualquer significado deve ser imposto por programas em nível de usuário. Tanto o Linux quanto o MS Windows usam essa abordagem.

## SEQUÊNCIA DE REGISTROS DE TAMANHO FIXO

É uma forma estruturada para o armazenamento de arquivos. Nela o arquivo é composto por uma série de registros com uma estrutura interna característica, e as operações de leitura/gravação trabalham com registros inteiros. No passado, quando o cartão de 80 colunas perfurado era utilizado, muitos sistemas operacionais de computadores de grande porte baseavam seus sistemas de arquivos em arquivos consistindo em registros de 80 caracteres. Esses sistemas também aceitavam arquivos com registros de 132 caracteres, destinados às impressoras de linha. Nenhum sistema de propósito geral atual utiliza esse modelo como seu sistema primário de arquivos.

## ÁRVORE DE REGISTROS

Uma organização que é composta por registros que não possuem necessariamente o mesmo tamanho, e cada um contém um campo com uma chave em uma posição fixa. A árvore é ordenada pelo campo chave de forma a permitir uma busca rápida. Esse tipo de arquivo é bastante diferente das sequências de bytes desestruturadas, sendo utilizado em alguns computadores de grande porte para o processamento de dados comerciais.



# MÉTODOS DE ACESSO

Os primeiros sistemas operacionais acessavam os registros de um arquivo na ordem em que eram gravados, possibilitando a gravação de novos registros apenas no final. A esse tipo de acesso dá-se o nome de **acesso sequencial**.

## ATENÇÃO

É possível retroceder registros, mas nunca pular ou lê-los fora de ordem.

Posteriormente, surgiu um método de acesso mais eficiente, o **acesso aleatório** (também conhecido como **acesso direto**), que permite a leitura/gravação de um registro diretamente na sua posição relativa ao início do arquivo. O acesso aleatório somente é possível quando o arquivo é definido com registros de tamanho fixo. Neste acesso, não existe restrição à ordem em que os registros são lidos ou gravados.

Em sistemas operacionais antigos, o tipo de acesso ao arquivo é determinado quando o arquivo é criado.



Nos sistemas operacionais modernos, todos os arquivos são de acesso aleatório, excetuando-se os casos em que o dispositivo no qual o arquivo está armazenado não permita esse tipo de acesso.

Existem ainda outros métodos de acesso menos comuns, como o **acesso indexado**, que envolve a construção de um índice para o arquivo. Esse índice contém ponteiros para os blocos e, para acessar uma posição de arquivo, primeiro pesquisa-se o índice e depois utiliza-se o ponteiro para a posição desejada.

# TIPOS DE ARQUIVOS

Os sistemas operacionais costumam suportar vários tipos de arquivos, sendo os mais comuns:

## ARQUIVOS REGULARES

Arquivos que contém informações genéricas como, por exemplo, dados dos usuários.

# DIRETÓRIOS

Arquivos de sistema usados para manter a estrutura do sistema de arquivos.

## ARQUIVOS ESPECIAIS DE CARACTERE

Relacionam-se com operações de E/S e costumam modelar dispositivos seriais.

## ARQUIVOS ESPECIAIS DE BLOCO

Usados para modelar dispositivos de bloco, em especial discos.

Em geral, arquivos regulares são classificados como **arquivo texto** ou **arquivo binário**.

### Arquivo texto

Um arquivo texto (ou arquivo ASCII) é constituído por linhas de texto que podem ter tamanhos diferentes e terminam por caracteres especiais para indicar o fim da linha. São arquivos que, quando exibidos na tela ou impressos, podem ser compreendidos pelas pessoas e, ainda, editados com um editor de textos comum.



### Arquivo binário

Arquivos binários não são arquivos texto. Sua listagem gera um conjunto de caracteres incompreensíveis. Eles podem ser arquivos de usuários (com ou sem estrutura interna) ou arquivos executáveis (com estrutura conhecida pelo sistema operacional e códigos que são executados pela UCP).

# DIRETÓRIOS

A estrutura de diretórios é o modo como o sistema organiza logicamente os diversos arquivos contidos em um disco. O **diretório** (ou pasta) é um arquivo que contém uma estrutura de dados com entradas associadas aos arquivos onde são armazenadas informações como localização física, nome, organização e demais atributos.

Quando é solicitada a abertura de um arquivo, o sistema operacional pesquisa o diretório até encontrar uma entrada com o nome do arquivo. Quando a entrada é localizada, o sistema operacional copia os atributos e os endereços de disco e os coloca em uma estrutura na memória, a **tabela de arquivos**, que contém todos os arquivos abertos. Quando o arquivo é fechado, sua entrada na tabela de arquivos é liberada.

A implementação mais simples de uma estrutura de diretórios é chamada de **nível único (um nível)**, no qual existe um único diretório contendo todos os arquivos do disco. Este modelo é bastante limitado, já que não permite que usuários criem arquivos com o mesmo nome, o que ocasionaria um conflito no acesso a eles.

## COMENTÁRIO

Este tipo de implementação não é mais utilizado atualmente.

Em outra estrutura, conhecida como diretório de **dois níveis**, existe um diretório para os arquivos do sistema e um diretório para cada usuário. Com esta implementação, cada usuário pode criar arquivos sem a preocupação de conhecer os demais arquivos do disco. Para que o sistema possa localizar arquivos nesta estrutura, existe um nível de diretório adicional denominado *master file directory*, indexado pelo nome do usuário. Nele, cada entrada aponta para o diretório de cada usuário.

## ATENÇÃO

Para referenciar um arquivo neste tipo de estrutura, é necessário especificar o diretório onde ele se encontra e o seu nome, especificando assim seu caminho (*path*).

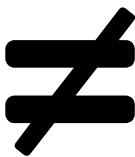
A organização dos arquivos em um único diretório não permite uma organização adequada. A extensão para um modelo de múltiplos níveis permite que os arquivos sejam mais bem organizados. Este modelo, chamado **estrutura de diretórios em árvore ou sistema de diretórios hierárquico**, é atualmente adotado pela maioria dos sistemas operacionais.

Na estrutura em árvore, é possível criar quantos diretórios se deseje, podendo um diretório conter arquivos ou outros diretórios (subdiretórios). Cada arquivo, nesta estrutura, possui um caminho (*path*) único que descreve todos os diretórios desde a raiz até o diretório no qual o arquivo está, mais o nome do arquivo.

Quando o sistema de arquivos é organizado como uma árvore de diretórios, os nomes de caminhos podem ser absolutos ou relativos.

Caminho absoluto

Consiste no caminho desde o diretório raiz (diretório inicial do sistema de arquivos) até o arquivo.



## Caminho relativo

É utilizado em conjunto com o conceito de diretório de trabalho (diretório atual), que é o diretório usado atualmente pelo processo e serve como base caso o nome do caminho não inicie com o diretório raiz. Quando é utilizado um caminho relativo, o caminho até o arquivo é buscado a partir do diretório de trabalho.

Cada processo possui seu próprio diretório de trabalho. Assim, se algum processo alterar seu diretório de trabalho, os outros processos do usuário não serão afetados.

A maioria dos sistemas que suporta estrutura de diretórios em árvore tem duas entradas especiais para cada diretório:

“.” → Diretório atual;

“..” → Diretório pai. Pode ser usada para subir na árvore de diretórios.

# IMPLEMENTAÇÃO DO SISTEMA DE ARQUIVOS

Os discos são o meio de armazenamento secundário mais comum no qual os arquivos são armazenados. As transferências de dados entre a memória e o disco são realizadas em unidades chamadas blocos, cada qual constituída por um ou mais setores.

## ★ EXEMPLO

Dependendo do disco, um setor pode variar de 32 a 4096 bytes, sendo mais comum setores de 512 bytes.

A maioria dos discos pode ser dividida em uma ou mais partições, com sistemas de arquivos independentes em cada partição. O Setor 0 do disco é chamado de **MBR** (Master Boot Record — registro mestre de inicialização) e é usado para inicializar o computador. No final do MBR está localizada a **tabela de partição**, que armazena os endereços de início e fim de cada partição. Uma das partições da tabela é marcada como ativa.

Quando o computador é inicializado, a **BIOS** lê e executa o MBR. A primeira coisa que o programa MBR faz é localizar a partição ativa, ler seu primeiro bloco (bloco de inicialização) e executá-lo. O programa no bloco de inicialização carrega o sistema operacional contido naquela partição. Cada partição começa com um bloco de inicialização, mesmo que ela não contenha um sistema operacional que possa ser inicializado.

## BIOS

BIOS (*Basic Input/Output System* - Sistema Básico de Entrada/Saída) é um pequeno programa gravado em uma memória não volátil usado para realizar a inicialização do hardware durante o processo de inicialização de um computador e para fornecer serviços de tempo de execução para sistemas operacionais e programas.

### ATENÇÃO

O esquema de uma partição de disco varia bastante entre sistemas de arquivos.

O primeiro item em uma partição costuma ser o **superbloco**. Ele contém todos os parâmetros chave a respeito do sistema de arquivos e é lido para a memória quando o computador é inicializado ou o sistema de arquivos é tocado pela primeira vez. Informações típicas no superbloco incluem um identificador para o tipo de sistema de arquivos, sua quantidade de blocos e outras informações administrativas.

Podem vir informações a respeito de blocos disponíveis no sistema de arquivos, na forma de um mapa de bits ou de uma lista encadeada.

Pode ser seguido pelos **i-nodes** (um arranjo de estruturas de dados, um por arquivo, dizendo tudo sobre ele). Depois pode vir o diretório-raiz, que contém o topo da árvore do sistema de arquivos. Por fim, o restante do disco contém todos os outros diretórios e arquivos.

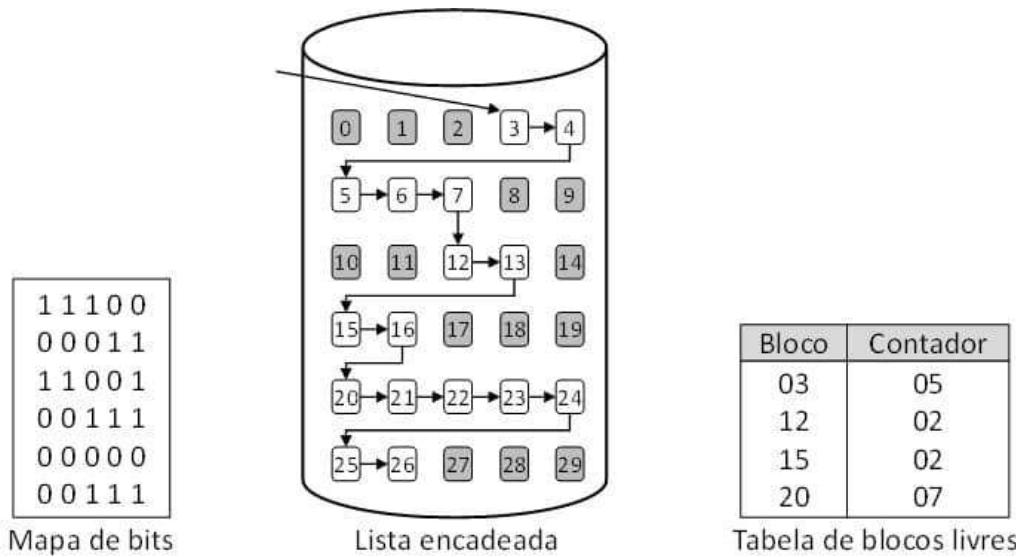
A criação de arquivos em disco exige que o sistema operacional tenha o controle de quais blocos no disco estão livres. Este controle é realizado através de uma estrutura de dados que armazena informações que possibilitam ao sistema de arquivos gerenciar o disco.

## DICA

A forma mais simples de implementar uma estrutura de espaços livres é através de uma tabela denominada mapa de bits. A cada entrada da tabela é associada um bloco do disco representado por um bit, que pode assumir valor igual a 0 (bloco livre) ou 1 (bloco alocado).

Uma segunda maneira de realizar este controle é por meio de uma lista encadeada de todos os blocos livres do disco. Cada bloco possui uma área reservada para armazenamento do endereço do próximo bloco, e a partir do primeiro bloco livre pode-se ter acesso aos demais de forma encadeada. Suas principais restrições são o espaço utilizado no bloco com informação de controle e o fato do algoritmo de busca de espaço livre sempre ter que realizar uma pesquisa sequencial na lista.

Outra solução leva em conta que blocos contíguos são geralmente alocados ou liberados simultaneamente, enxergando o disco como um conjunto de segmentos de blocos livres. Assim, é possível manter uma tabela de blocos livres com o endereço do primeiro bloco de cada segmento e o número de blocos livres contíguos que se seguem.



## ALOCAÇÃO CONTÍGUA

A alocação contígua consiste em armazenar um arquivo em blocos contíguos de dados no disco. Neste tipo de alocação, o sistema localiza um arquivo através do endereço do primeiro bloco e da sua quantidade de blocos.

A alocação de espaço de disco contíguo tem duas vantagens.

Primeira vantagem

Ela é simples de implementar. Dado o número do primeiro bloco, o número de qualquer outro bloco pode ser encontrado mediante uma simples adição.

### Segunda vantagem

O desempenho da leitura é excelente, pois o arquivo inteiro pode ser lido do disco em uma única operação.

O acesso a arquivos dispostos contiguamente no disco é bastante simples tanto para o acesso sequencial quanto para o acesso aleatório. Seu principal problema é a alocação de espaço livre para novos arquivos, já que para um arquivo ser criado com  $n$  blocos é necessário que exista uma cadeia de  $n$  blocos dispostos sequencialmente no disco.

O disco pode ser visto como um grande vetor no qual os elementos podem ser considerados segmentos com tamanhos diferentes de blocos contíguos, dispostos alternadamente entre segmentos ocupados e segmentos livres. No momento em que o sistema operacional deseja alocar espaço para armazenar um novo arquivo, pode existir mais de um segmento livre disponível com o tamanho exigido.

### ATENÇÃO

Neste caso, é necessário que alguma estratégia de alocação seja adotada para selecionar qual o segmento na lista de blocos livres deve ser escolhido.

Seja qual for a estratégia adotada, se o bloco livre selecionado para o armazenamento do arquivo for maior que o tamanho do arquivo, sobrará um espaço livre ao final do bloco. Com a operação em longo prazo, surgirão vários blocos livres espalhados pelo disco, ocorrendo a chamada **fragmentação do disco**.

Quando o disco estiver muito fragmentado para que se possa alocar espaço para a criação do arquivo, será necessário realizar a **desfragmentação do disco**, que consiste em mover os arquivos para abrir espaço suficiente para o novo arquivo.

### DICAS

A desfragmentação é um processo demorado que deve ser realizado periodicamente.

Atualmente, a alocação contígua é usada em CD-ROMs. Todos os tamanhos de arquivos são conhecidos antecipadamente e jamais mudarão durante o uso subsequente do sistema de arquivos do

## ALOCAÇÃO POR LISTA ENCADEADA

Na alocação por lista encadeada, um arquivo é organizado como um conjunto de blocos ligados logicamente, independente da sua localização física. Cada bloco deve possuir um ponteiro para o bloco seguinte e assim por diante.

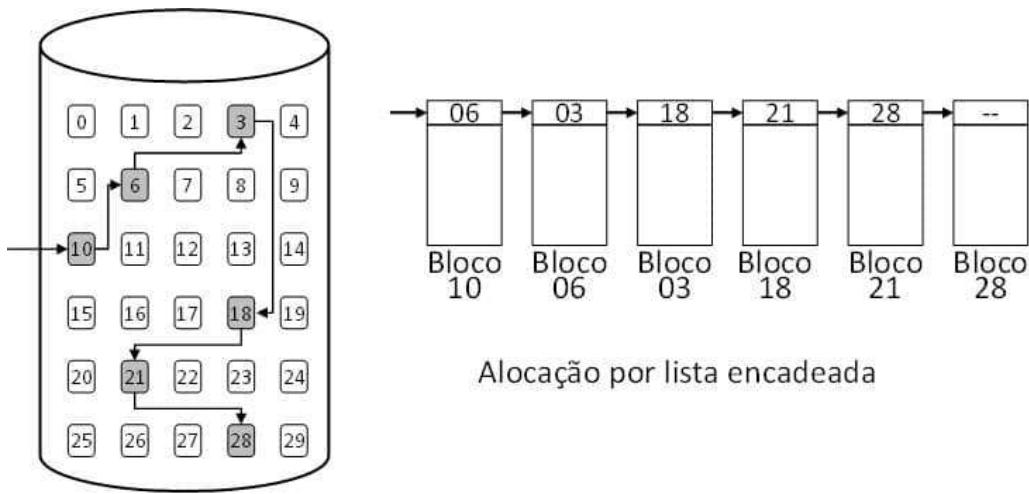
### ATENÇÃO

A entrada de diretório precisa armazenar somente o endereço do primeiro bloco.

A fragmentação do disco não ocorre na alocação encadeada já que os blocos alocados para determinado arquivo não precisam estar em posições contíguas. O que ocorre, neste método, é a **fragmentação de arquivos**, que é a quebra do arquivo em diversos blocos espalhados pelo disco. Tal fragmentação resulta no aumento do tempo de acesso, pois o processo de leitura/gravação provoca muitos deslocamentos da cabeça de leitura/gravação do disco.

### DICA

Para otimizar o tempo das operações de E/S é importante que o sistema de arquivos seja desfragmentado periodicamente.



A alocação por lista encadeada permite apenas o acesso sequencial aos blocos de um arquivo. Isso constitui uma das principais desvantagens dessa técnica. Além disso, é desperdiçado espaço nos blocos com o armazenamento de ponteiros.

Outro problema da alocação por lista encadeada é a confiabilidade. Se o valor de um ponteiro é corrompido, se perde o encadeamento do arquivo.

## ALOCAÇÃO POR LISTA ENCADEADA UTILIZANDO ÍNDICE

É um esquema de alocação muito parecido com a alocação por lista encadeada, mas no lugar de fazer o encadeamento utilizando um ponteiro no bloco, o encadeamento é mantido em uma tabela.

Embora a cadeia ainda precise ser seguida para o acesso aleatório, ela é seguida por intermédio da tabela e não consultando bloco a bloco. Se a tabela for mantida em memória, o acesso torna-se bem mais rápido.

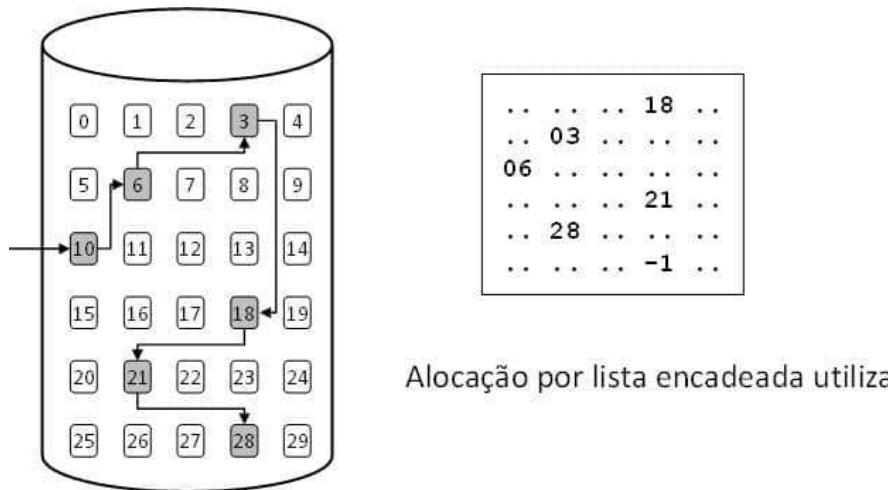
Essa tabela na memória principal é chamada de **FAT** (*File Allocation Table* — tabela de alocação de arquivos). Usando essa organização, todo o bloco do disco fica disponível para os dados do arquivo.

Da mesma maneira que no método anterior, é suficiente para a entrada de diretório manter um único inteiro (o número do bloco inicial) e ainda assim ser capaz de localizar todos os blocos, não importando o tamanho do arquivo.

A principal desvantagem é que a tabela pode ser muito grande para discos grandes. Esse problema pode ser minimizado agrupando os blocos em **clusters** para formar unidades maiores, mas isto provoca um maior desperdício por causa da parte não utilizada do último *cluster*, que tenderá a ser maior.

# CLUSTERS

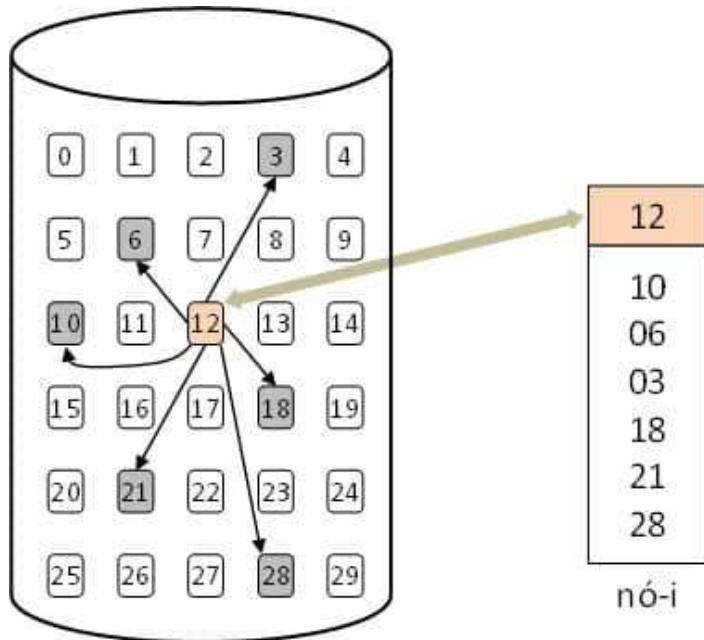
Agrupamentos de blocos de um disco.



Alocação por lista encadeada utilizando índice

## I-NODES (ALOCAÇÃO INDEXADA)

O método consiste em associar a cada arquivo uma tabela denominada **i-node** (nó-índice ou nó-i), que lista os atributos e os endereços em disco dos blocos do arquivo.



A grande vantagem desse esquema sobre a alocação por lista encadeada utilizando índice é que o i-node precisa estar na memória apenas quando o arquivo correspondente estiver aberto.

Se cada i-node ocupar  $n$  bytes e  $k$  arquivos estiverem abertos simultaneamente, a memória total ocupada pelo arranjo contendo os i-nodes para os arquivos abertos é de apenas  $k \times n$  bytes.

Esse arranjo é, em geral, muito menor do que o espaço ocupado pela tabela de arquivos na alocação por lista encadeada utilizando índice.

Os primeiros endereços de disco são armazenados no próprio i-node. Isto significa que para arquivos pequenos toda a informação está contida no próprio i-node e pode ser transferida para a memória quando o arquivo é aberto.

## BLOCO INDIRETO SIMPLES

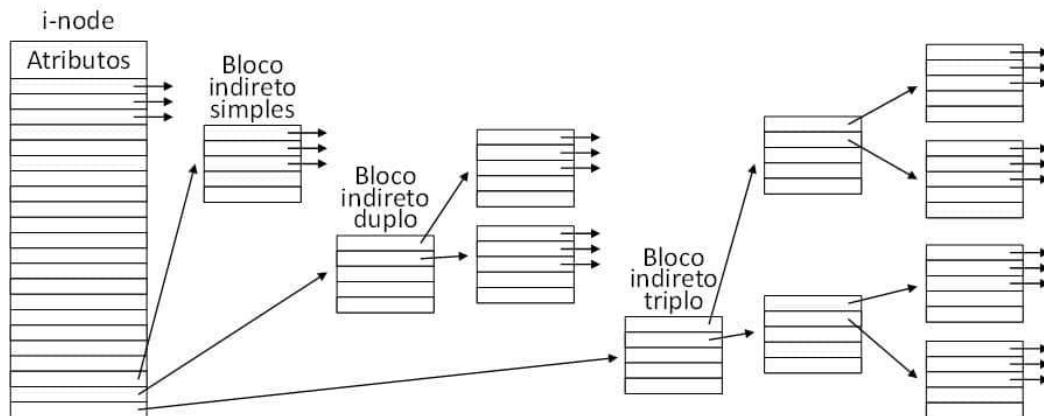
Contém endereços de blocos adicionais para o arquivo. É usado para arquivos maiores.

## BLOCO INDIRETO DUPLO

Contém o endereço do bloco que possui uma lista de blocos indiretos simples relativos ao arquivo.

## BLOCO INDIRETO TRIPLO

Contém o endereço do bloco que possui uma lista de blocos indiretos duplos.



## IMPLEMENTAÇÃO DE CACHE

O acesso a disco é lento quando comparado com o acesso à memória principal. Este é o fator básico para as operações de E/S com discos serem um problema para o desempenho do sistema. Com o objetivo de minimizar tal problema, a maioria dos sistemas de arquivos implementa uma técnica denominada **cache**.

Neste esquema, o sistema operacional reserva uma área na memória principal para que se tornem disponíveis caches utilizados em operações de acesso ao disco.

Quando uma operação de E/S é realizada, o sistema verifica se a informação desejada se encontra na cache. Caso esteja disponível, não é necessário acesso ao disco. Caso o bloco requisitado não se

encontre na cache, a operação de E/S é realizada e a cache é atualizada. Como existe uma limitação no tamanho da cache, o sistema deve adotar uma política para substituição de blocos.

Apesar de melhorar o desempenho do sistema, aspectos de segurança devem ser levados em consideração. No caso de blocos de dados permanecerem por muito tempo na cache, a ocorrência de problemas de energia pode ocasionar a perda de tarefas já realizadas e consideradas salvas em disco.

Existem duas maneiras de tratar este problema.

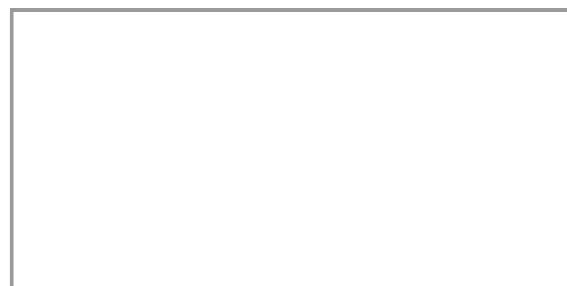
No primeiro caso, o sistema operacional possui uma rotina que executa periodicamente atualizando em disco todos os blocos modificados da cache.

Uma segunda alternativa, conhecida como *write-through*, é realizar imediatamente uma atualização no disco sempre que um bloco da cache for modificado.

Quando a memória cache não atualiza o disco imediatamente, temos a chamada cache *write-back*. A técnica *write-back* implica em menor quantidade de operações de E/S, porém o risco de perda de dados é maior. Isso não acontece nas caches *write-through* em função do seu próprio funcionamento, mas o aumento considerável nas operações de E/S torna este método menos eficiente.

## A EVOLUÇÃO DOS SISTEMAS DE ARQUIVOS E DAS MÍDIAS DE ARMAZENAMENTO

Assista, a seguir, como os sistemas de arquivos evoluíram com o tempo:



## VERIFICANDO O APRENDIZADO

## MÓDULO 2

# O SISTEMA DE ARQUIVOS DO LINUX

O Linux trata o conceito de arquivo de uma forma bastante ampla. Um arquivo não precisa ser um objeto em um disco, mas pode ser qualquer objeto capaz de manipular dados. O Linux manipula todos esses objetos ocultando detalhes específicos de cada implementação por intermédio de uma camada de software denominada **sistema de arquivos virtual** (*Virtual File System* – VFS).

O VFS define quatro tipos de objetos principais:

## I-NODE

Representa um arquivo individual.

## ARQUIVO

Representa um arquivo aberto.

## SUPERBLOCO

Representa um sistema de arquivos inteiro.

## DENTRY

Representa uma entrada de diretório individual.

Para cada um desses objetos, o VFS define um conjunto de chamadas de sistema, e cada objeto possui um ponteiro para uma tabela de funções que lista as funções reais que implementam as chamadas de sistema dos objetos. O VFS pode executar uma operação sobre um dos objetos do sistema de arquivos, chamando a função apropriada na tabela de funções sem precisar saber de antemão com que tipo de objeto está lidando.

Os objetos *arquivo* pertencem a um único processo, mas os objetos i-node não. Há um objeto *arquivo* para cada instância de um arquivo aberto, mas um único objeto i-node.

## ATENÇÃO

Mesmo quando um arquivo não está sendo mais utilizado por algum processo, seu objeto i-node ainda pode ser armazenado em cache pelo VFS.

Arquivos de diretório são manipulados de forma diferente de outros arquivos. O Linux define as chamadas de sistema para diretórios (criação, exclusão, renomeação de arquivo etc.) no objeto i-node, no lugar do objeto *arquivo*.

O Linux mantém um único objeto *superbloco* para cada dispositivo de disco montado e para cada sistema de arquivos de rede conectado. A principal função do objeto *superbloco* é dar acesso a i-nodes.

Um objeto *dentry* representa uma entrada de diretório que pode ser o nome de um diretório no nome de caminho de um arquivo.

## ★ EXEMPLO

arquivo “/home/maria/teste.txt” possui as entradas de diretório “/”, “home”, “maria” e “teste.txt”, sendo cada uma destas entradas representada por um objeto *dentry* diferente.

A primeira versão do Linux implementava o sistema de arquivos do **MINIX**, que limitava os nomes de arquivos a 14 caracteres e tinha um limite de 64 MB para o tamanho máximo que eles poderiam ter.

## MINIX

Sistema Operacional Unix-like escrito por Andrew S. Tanenbaum e utilizado por Linus Torvalds como base para o desenvolvimento do Linux.

Desde o início, havia o interesse em desenvolver um sistema de arquivos mais satisfatório, que foi implementado com o sistema de arquivos **ext**, o qual permitia arquivos com até 2 GB de dados e com nomes de até 255 caracteres. Seu problema consistia em ser mais lento que o sistema de arquivos do MINIX.

## ❓ VOCÊ SABIA

A versão 2 do **ext**, conhecida como **ext2**, resolveu os problemas de desempenho do **ext**, permitindo nomes longos e tornando o principal sistema de arquivos do Linux.

Devido à implementação do VFS, o Linux suporta dezenas de sistemas de arquivos diferentes. Quando o sistema é iniciado, o conjunto de funções para o acesso ao sistema de arquivos principal deve estar compilado no núcleo do Linux para que possa ser montado e utilizado. Durante a execução, módulos para acesso a outros sistemas de arquivos podem ser carregados dinamicamente, permitindo, assim, a utilização simultânea de diferentes sistemas de arquivos.

O Linux não exige que arquivos possuam extensões (caracteres após um caractere “.”), nem limita o tipo ou a quantidade de caracteres presentes em uma extensão. Ainda, arquivos podem ter qualquer quantidade de extensões (o que é comum). No entanto, vários programas esperam que arquivos possuam extensões.

Está a cargo dos programas interpretar ou não o tipo do arquivo por sua extensão, visto que para o Linux a extensão de um arquivo não possui nenhum significado.

O diretório raiz é chamado /, que além de ser utilizado para separar nomes de diretórios, é um caractere que contém subdiretórios. Sob o diretório raiz, existe um grupo de diretórios comuns à maioria das distribuições Linux. Alguns dos que estão localizados diretamente no diretório raiz, são:

/bin → Arquivos executáveis;

/boot → Arquivos de configuração do boot, kernel e outros arquivos necessários para a inicialização do sistema;

/dev → Contém os dispositivos do sistema;

/etc → Arquivos de configuração, scripts de inicialização etc;

/home → Diretórios home para usuários do sistema;

/lib → Bibliotecas e módulos do sistema;

/lib64 → Semelhante ao /lib, mas contém arquivos específicos do sistema 64 bits;

/media → Onde são montados dispositivos externos;

/mnt → Local para montagem manual de dispositivos. Normalmente utilizado para montagens provisórias;

/opt → Fornece um local opcional para aplicações serem instaladas. Normalmente utilizado para instalação de aplicativos que não fazem parte do sistema;

/proc → Diretório dinâmico especial que mantém informação sobre o estado do sistema, incluindo os processos em execução. Contém algumas variáveis do kernel que podem, inclusive, ser modificadas;

/root → Diretório home do usuário root (administrador do sistema);

/sbin → Arquivos executáveis para administração do sistema;

/srv → Contém arquivos de alguns serviços do sistema. Por exemplo, servidor web;

/sys → Semelhante ao /proc. Contém arquivos especiais do kernel;

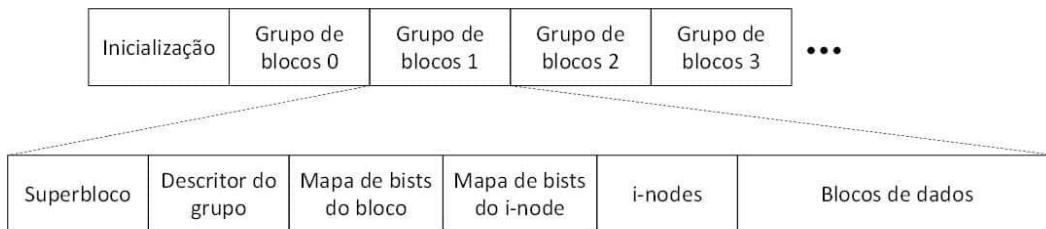
/tmp → Contém arquivos temporários;

/usr → Aplicativos e arquivos que são, na maioria das vezes, disponíveis para acesso por todos os usuários;

/var → Arquivos de logs e bancos de dados.

## EXT2

O ext2 foi um dos sistemas de arquivos mais populares desenvolvido para o Linux. O layout de uma partição ext2 é mostrado na figura a seguir:



### Layout de uma partição ext2.

O primeiro bloco (bloco de inicialização) não é utilizado pela partição, é reservado para a inicialização do computador. Após o primeiro bloco seguem grupos de blocos de mesma organização.

O **superbloco** contém informações sobre o layout do sistema de arquivos, como a quantidade de i-nodes e a quantidade de blocos de disco. O **descritor do grupo** contém informações sobre a localização dos **mapas de bits**, a quantidade de blocos livres e i-nodes no grupo, e a quantidade de

diretórios no grupo. Dois mapas de bits são usados para controlar os blocos livres e i-nodes livres. Cada mapa contém um bloco de comprimento. Em seguida, estão os i-nodes, os quais são numerados de 1 até algum máximo. Cada i-node tem 128 bytes de comprimento e descreve exatamente um arquivo.

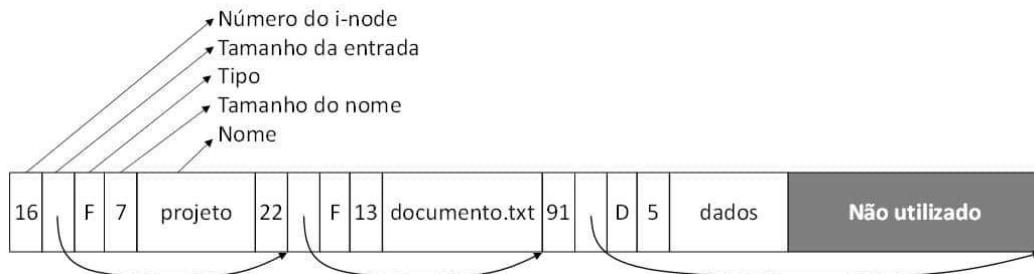
## ATENÇÃO

Um i-node contém informações de contabilidade e dados suficientes para localizar todos os blocos de disco que contêm os dados do arquivo.

Por fim, estão os **blocos de dados**, onde os arquivos e diretórios estão armazenados.

Os diretórios ficam dispersos pelos grupos de blocos do disco. O *ext2* procura colocar arquivos de dados no mesmo grupo de blocos que o i-node do arquivo original. Os mapas de bits são usados para tomar decisões rápidas sobre onde alocar novos dados do sistema de arquivos. Quando novos blocos de arquivos são alocados, o *ext2* pré-aloca 8 blocos adicionais de maneira a minimizar a fragmentação de arquivos por futuras operações de escrita. Essa pré-alocação equilibra a carga do sistema de arquivos.

Um diretório armazena nomes de arquivos, sendo ilustrado na figura a seguir, onde existem entradas para 2 arquivos (projeto e documento.txt) e um diretório (dados).



### Diretório do *ext2*.

Dentro de um diretório, as entradas para arquivos e diretórios estão fora de ordem. Pode acontecer de uma entrada não ocupar blocos de disco inteiros, então é comum haver bytes não utilizados no final de cada bloco de disco.

Cada entrada de diretório consiste em 4 campos de comprimento fixo e 1 campo de comprimento variável.

Primeiro campo

O primeiro campo é o número do i-node (16 para projeto, 22 para documento.txt e 91 para dados).

Segundo campo

O segundo campo contém o tamanho da entrada, incluindo espaços não utilizados no seu final.

### Terceiro campo

O terceiro campo identifica o tipo (arquivo, diretório, link etc.).

### Quarto campo

O quarto campo é o tamanho do nome do arquivo.

### Quinto campo

O quinto campo possui tamanho variável e armazena o nome do arquivo.

Permissões
Contador de links
UID
GID
Tamanho do arquivo
Horários
Endereços dos 12 primeiros blocos do disco
Bloco indireto simples
Bloco indireto duplo
Bloco indireto triplo

### Formato do i-node.

Cada i-node do *ext2* possui o formato ilustrado acima, em que:

Permissões → Permissões de acesso ao arquivo;

Contador de links → Quantidade de **hardlinks** que o arquivo possui;

UID → Dono do arquivo;

GID → Grupo do dono do arquivo.

## HARDLINKS

*Hardlinks* são diferentes entradas de diretórios que apontam para o mesmo i-node (o mesmo arquivo).

# JOURNALING

Para evitar perda de dados após problemas, como falta de energia, é necessário escrever blocos de dados no disco tão logo o bloco seja criado ou alterado, levando a um baixo desempenho do sistema devido à necessidade de movimentação das cabeças de leitura/gravação dos discos.

## SAIBA MAIS

Para minimizar tal problema, foi desenvolvido um recurso conhecido como **journaling**, no qual as alterações são gravadas sequencialmente em um diário em vez de serem escritas diretamente nos blocos de disco.

Um conjunto de operações que executam uma tarefa específica é denominada **transação**. Quando uma transação é escrita no diário, ela é considerada confirmada e o sistema pode prosseguir.

A seguir, as entradas do diário relacionadas com as transações são reexecutadas nas estruturas reais do sistema de arquivos. Quando uma transação é concluída (totalmente salva no disco), ela é removida do diário, que pode estar em uma seção separada do sistema de arquivos ou em um eixo separado do disco. Como fica em uma seção confinada do disco, é mais eficiente por diminuir os tempos de disputa e busca do cabeçote.

## ATENÇÃO

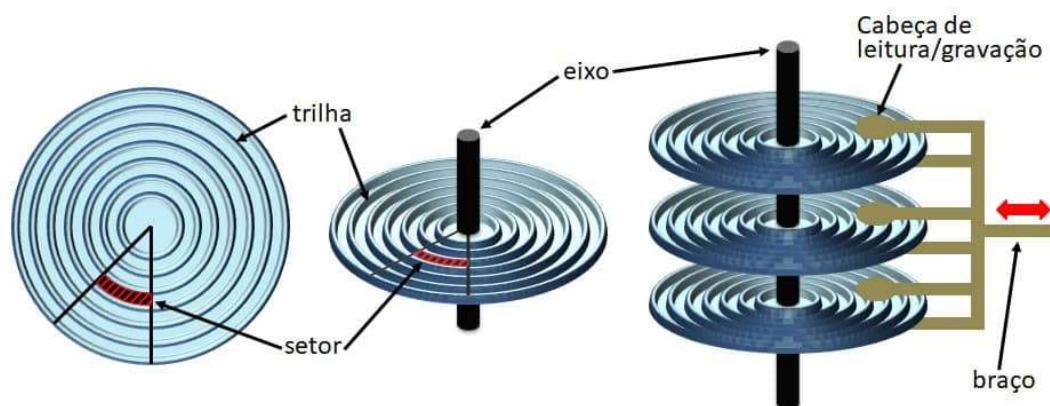
Se o sistema cair e algumas transações permanecerem no diário, elas deverão ser concluídas quando o sistema se recuperar.

O journaling foi implementado no Linux na terceira versão do *ext*, conhecida como **ext3**.

# DISCO RÍGIDO

Um disco rígido é constituído por **discos** sobrepostos, unidos por um **eixo** vertical girando a uma velocidade constante. Cada disco é composto por **trilhas** concêntricas, que são divididas em **setores**. As trilhas dos diferentes discos que ocupam a mesma posição vertical formam um **cilindro**. Para cada

superfície de um disco existe uma **cabeça** de leitura/gravação. O conjunto de cabeças é preso a um **braço** que se movimenta entre os vários cilindros no sentido radial.



O tempo necessário para ler/gravar um bloco de dados é função de três fatores: tempo de busca, latência e transferência.

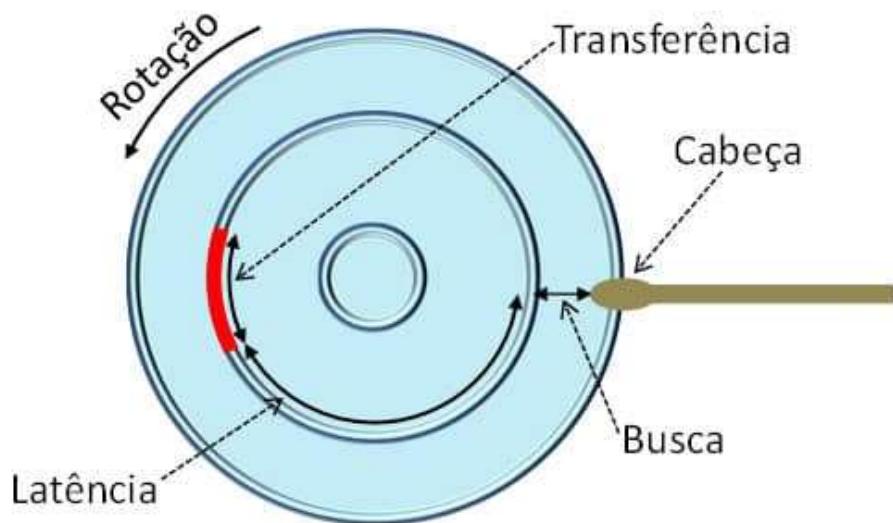
O **tempo de busca** (seek) é o tempo gasto para mover o braço até o cilindro onde o bloco se encontra.



O **tempo de latência** é o tempo de espera até que o setor desejado se posicione sob a cabeça de leitura/gravação.



O **tempo de transferência** corresponde ao tempo necessário para ler/gravar o bloco.



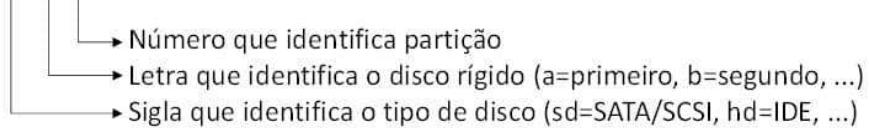
## PARTIÇÕES

O particionamento de disco é uma divisão de seu espaço disponível em seções de forma que possam ser utilizadas de maneira independente. Uma partição pode ocupar um disco inteiro, ou podem ser criadas várias partições por disco, cada uma contendo um sistema de arquivos independente.

As informações sobre as partições existentes em um disco são gravadas em uma tabela de partição no próprio disco. Para ser utilizada, uma partição deve ser formatada com um sistema de arquivos para que estes possam ser gravados nela.

No Linux, os dispositivos são identificados por arquivos especiais que ficam armazenados no diretório `/dev`. A identificação de discos e partições é realizada de acordo com o seguinte padrão:

**/dev/sda1**



Algumas identificações de discos e partições em sistemas Linux:

`/dev/fd0` → Primeira unidade de disquetes;

`/dev/fd1` → Segunda unidade de disquetes;

`/dev/sda` → Primeiro disco rígido na primeira controladora SATA ou SCSI;

`/dev/sda1` → Primeira partição do primeiro disco rígido SATA ou SCSI;

`/dev/sda2` → Segunda partição do primeiro disco rígido SATA ou SCSI;

`/dev/sdb` → Segundo disco rígido na primeira controladora SATA ou SCSI;

`/dev/sdb3` → Terceira partição do segundo disco rígido SATA ou SCSI;

`/dev/sr0` → Primeiro CD-ROM SATA ou SCSI;

`/dev/hda` → Primeiro disco rígido na primeira controladora IDE;

`/dev/hda1` → Primeira partição do primeiro disco rígido IDE.

Clique [aqui](#) para fazer o download de um documento com exemplo de criação de partições em disco utilizando o Linux.

# FORMATAÇÃO

O processo de criação de partições em um disco reserva espaço para o sistema de arquivos, mas ainda não faz com que o dispositivo possa ser utilizado. A fim de que seja possível fazer uso das partições criadas, é necessário criar uma estrutura para o sistema de arquivos. É um processo conhecido como **formatação**.

Formatar uma partição Linux significa criar os grupos de blocos, cada um deles contendo seu superbloco, descritor do grupo, mapas de bits, i-nodes e reservar espaço para os blocos de dados.

Clique [aqui](#) para fazer o download de um documento com exemplo de formatação de partições em disco utilizando o Linux.

# MONTAGEM DO SISTEMA DE ARQUIVOS

Muitos sistemas possuem dois ou mais discos ou partições. Além disso, muitos sistemas atuais permitem a utilização de diversos dispositivos de armazenamento simultaneamente, como discos ópticos e unidades USB, devendo se pensar em uma forma de acessar os diferentes sistemas de arquivos.

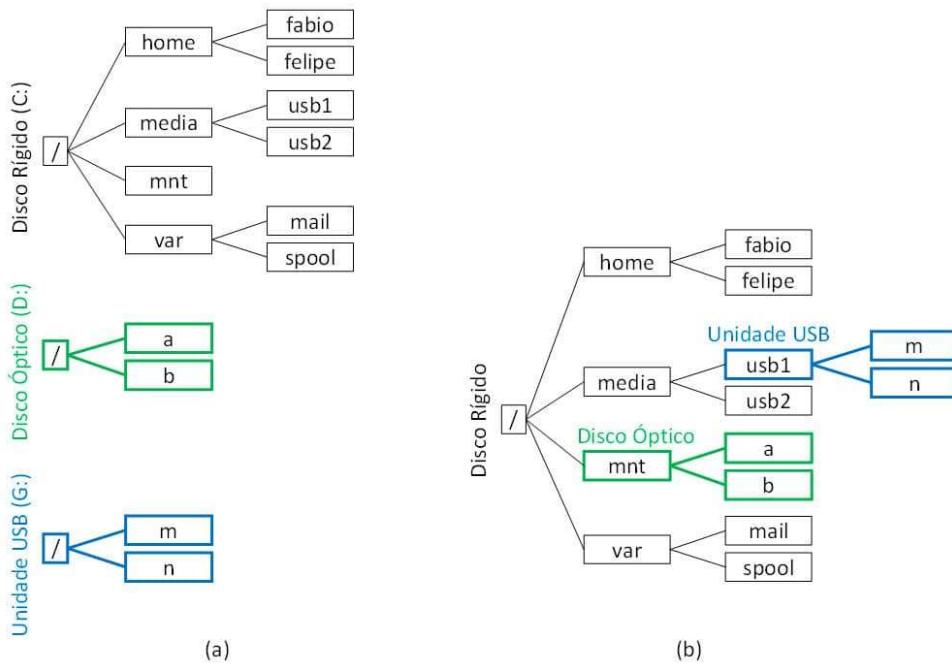
Uma solução está em manter cada sistema separado e encontrar uma forma de referenciá-los, conforme a maneira adotada pela Microsoft para o sistema operacional Windows. Nela, os dispositivos são identificados por uma letra seguido pelo caractere dois pontos.

## ★ EXEMPLO

Uma partição de um disco rígido pode ser identificada por **C:**, uma unidade óptica por **D:**, um dispositivo USB por **G:**, e assim por diante. Cada unidade possui seu próprio diretório raiz, arquivos e subdiretórios. Nessa solução, o usuário precisa especificar tanto o dispositivo quanto o arquivo.

Outro meio, adotado pelo Linux, é fazer com que o sistema de arquivos de um dispositivo/partição seja montado sobre a árvore de diretórios do sistema. Dessa forma, uma unidade ótica poderia ser montada, por exemplo, no diretório **/mnt**, enquanto um dispositivo USB poderia ser montado no diretório **/media/usb1**.

A figura abaixo mostra a estrutura do sistema de arquivos descrito (a) no Microsoft Windows e (b) no Linux.



🕒 Exemplo de montagem de dispositivos (a) no MS Windows e (b) no Linux.

Clique [aqui](#) para fazer o download de um documento com exemplo de montagem e desmontagem de partições em disco utilizando o Linux.

## OUTROS COMANDOS PARA GERENCIAMENTO DE PARTIÇÕES

Além dos comandos vistos nos exemplos para criação e gerenciamento de partições, o Linux possui outros utilitários que podem ser úteis para a administração do sistema. Vejamos alguns destes comandos.

### FSCK

Utilitário utilizado para verificar e, se necessário, corrigir um sistema de arquivos. Se não for informado o tipo do sistema de arquivos, o *fsck* procurará por essa informação no arquivo */etc/fstab*.

Algumas opções:

-t → Especifica o tipo do sistema de arquivos;

-C → Mostra uma barra de progresso da verificação. Não está disponível para todos os sistemas de arquivos;

-V → Gera saída detalhada das operações realizadas.

Exemplo:

```
$ sudo fsck -t ext4 /dev/sdb1
fsck de util-linux 2.34
e2fsck 1.45.5 (07-Jan-2020)
/dev/sdb1: limpo, 11/196608 ficheiros, 31036/786432 blocos
```

## DF

Utilitário para exibição do espaço livre/ocupado por cada sistema de arquivos montados. Informa a partição onde o sistema de arquivos reside, o tamanho do sistema de arquivos, os espaços utilizado e disponível, o percentual de uso e onde se localiza o ponto de montagem do sistema de arquivos.

Algumas opções:

-h → Mostra o espaço livre/ocupado em MB, KB, GB em vez de blocos;

-k → Lista em Kbytes;

-l → Somente lista sistema de arquivos locais;

-m → Lista em Mbytes.

Exemplo:

```
$ df -h
```

Sist. Arq.	Tam.	Usado	Disp.	Uso%	Montado em
udev	967M	0	967M	0%	/dev
tmpfs	199M	1,3M	198M	1%	/run
/dev/sda5	15G	6,7G	6,9G	50%	/
Compartilhada	446G	430G	17G	97%	/media/sf_Compactada

## LSBLK

Utilitário que lista informações sobre os dispositivos de bloco do sistema, excetuando-se os discos de RAM. As informações incluem o tamanho total da partição/bloco e o ponto de montagem, se o sistema de arquivos estiver montado. Este comando não informa o espaço em disco utilizado/livre nos sistemas de arquivos.

Algumas opções:

-a → Lista todos os dispositivos, incluindo discos de RAM;

-b → Informa o tamanho em bytes;

-f → Exibe informações sobre os sistemas e arquivos.

Exemplo:

\$ **lsblk**

```
NAME  MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda    8:0    0   15G  0 disk 
|---sda1  8:1    0   512M 0 part /boot/efi
|---sda2  8:2    0     1K 0 part 
└---sda5  8:5    0  14,5G 0 part /
sdb    8:16   0   10G  0 disk 
|---sdb1  8:17   0     3G 0 part 
|---sdb2  8:18   0     2G 0 part /extra/b2
|---sdb3  8:19   0     1G 0 part 
└---sdb4  8:20   0     4G 0 part 
sr0    11:0   1 1024M 0 rom
```

## **/ETC/FSTAB**

Arquivo que contém informações sobre os sistemas de arquivos a serem utilizados no sistema, como a partição em que se encontra, o tipo do sistema de arquivos, o ponto de montagem etc.

# **PREPARANDO UM DISCO RÍGIDO PARA A UTILIZAÇÃO NO LINUX**

Veja, a seguir, como o disco rígido deve ser preparado para ser utilizado no Linux:



# VERIFICANDO O APRENDIZADO

## MÓDULO 3

- 
- Empregar as ferramentas de gerenciamentos de arquivos do Linux

## CONCEITOS

O Linux é um sistema operacional, ou melhor, o núcleo do sistema. É o responsável pelo gerenciamento dos dispositivos e recursos do sistema, oferecendo para os softwares que executam acima dele chamadas de sistema que transformam o verdadeiro hardware em uma máquina mais fácil de ser utilizada.

O Linux é a base, mas ainda faltam softwares que forneçam a funcionalidade que os usuários desejam de um sistema. A fim de oferecer esta funcionalidade, organizações e pessoas empacotam um conjunto de softwares que, com o núcleo do sistema (o Linux), formam um sistema completo que pode ser instalado e utilizado pelas pessoas.

### SAIBA MAIS

Esse conjunto de softwares é conhecido como **distribuição**.

Cada distribuição é livre para escolher os pacotes de software que serão instalados, e isso se torna mais evidente com relação ao ambiente gráfico. Existem várias interfaces gráficas que podem ser utilizadas, cada uma configurada com diferentes softwares.

### EXEMPLO

Algumas das interfaces gráficas mais populares para o Linux são: GNOME, KDE, Cinnamon, MATE e XFCE.

O Linux também possui uma série de comandos em modo texto que permitem realizar praticamente qualquer tarefa no sistema, e, ao contrário das interfaces gráficas, são utilitários padrão (ou quase isso) disponíveis para utilização em qualquer distribuição. Por essa razão, estudaremos os comandos em modo texto, que permitirão que você possa utilizar qualquer sistema Linux.

Durante a inicialização, o Linux monta um sistema de arquivos conhecido como **sistema de arquivos raiz**. Ele tem início em um diretório representado pela barra de divisão “/”, que recebe o nome de **root** (raiz).

## ATENÇÃO

Não confundir o diretório root com o usuário root (administrador do sistema).

Os demais sistemas de arquivos que forem necessários para a execução do sistema serão montados em pontos de montagem da hierarquia de diretórios.

## SAIBA MAIS

Para alguns sistemas operacionais, como o Microsoft Windows, um arquivo torna-se oculto quando é ativada uma de suas propriedades. Ou seja, ser oculto é uma propriedade.

O Linux trata arquivos ocultos de uma forma diferente. Um arquivo oculto nesse sistema operacional é aquele que inicia com o caractere ponto “.”. Esse tipo de arquivo não aparece em listagens do conteúdo dos diretórios, a não ser que seja explicitamente solicitado.

# COMANDOS PARA MANIPULAÇÃO DE DIRETÓRIOS

## LS

Lista o conteúdo do diretório corrente.

## Sintaxe: **ls [opções] [caminho]**

Se *caminho* for informado, mostra o conteúdo do diretório *caminho*, senão mostra o conteúdo do diretório corrente.

Algumas opções:

**-a** → Lista todos os arquivos de um diretório, incluindo arquivos ocultos;

**-h** → Mostra o tamanho dos arquivos em Kbytes, Mbytes, Gbytes;

**-i** → Mostra o número do i-node de cada arquivo.

**-l** → Mostra mais informações, como lista as permissões, data de modificação, donos, grupos etc.

**-R** → Lista diretórios e subdiretórios recursivamente.

**-r** → Inverte a ordem de classificação.

**-c** → Classifica pela data de alteração.

**--help** → Mostra uma tela de ajuda.

Exemplo:

```
fabio@ubuntu20:~$ ls
```

```
'Área de Trabalho' Downloads Modelos Público teste Vídeos Documentos Imagens Música snap teste.txt
```

```
fabio@ubuntu20:~$ ls -a
```

```
.           .gnupg      .ssh  
..          Imagens     .sudo_as_admin_successful  
'Área de Trabalho'   .joe_state   teste  
.bash_history    .lesshst    teste.txt  
.bash_logout     .local      .vboxclient-clipboard.pid  
.bashrc         Modelos     .vboxclient-display-svga-x11.pid  
.cache          Música      .vboxclient-draganddrop.pid  
.config         .profile     .vboxclient-seamless.pid  
Documentos       Público     Vídeos  
Downloads        snap
```

```
fabio@ubuntu20:~$ ls -lh
```

```
total 20M
```

```
drwxr-xr-x 2 fabio fabio 4,0K ago 9 20:18 'Área de Trabalho'  
drwxr-xr-x 2 fabio fabio 4,0K ago 9 20:18 Documentos  
drwxr-xr-x 2 fabio fabio 4,0K ago 9 20:18 Downloads  
drwxr-xr-x 2 fabio fabio 4,0K ago 9 20:18 Imagens  
drwxr-xr-x 2 fabio fabio 4,0K ago 9 20:18 Modelos  
drwxr-xr-x 2 fabio fabio 4,0K ago 9 20:18 Música  
drwxr-xr-x 2 fabio fabio 4,0K ago 9 20:18 Público  
-rwxr-xr-x 1 fabio fabio 20M ago 29 14:30 snap  
drwxrwxr-x 2 fabio fabio 4,0K ago 14 10:14 teste  
-rw-rw-r-- 1 fabio fabio 5,0K ago 13 17:21 teste.txt  
drwxr-xr-x 2 fabio fabio 4,0K ago 9 20:18 Vídeos
```

## PWD

Mostra o caminho do diretório de trabalho.

Exemplo:

```
$ pwd
```

```
/home/fabio
```

No exemplo, o comando *pwd* mostrou que o diretório de trabalho é o */home/fabio*, ou seja, o diretório padrão de trabalho do usuário *fabio* (diretório *home* de *fabio*).

## CD

Altera o diretório atual de trabalho.

Sintaxe: **cd [diretório]**

Altera o diretório de trabalho para o passado como parâmetro. Se o diretório de destino não for especificado, vai para o diretório *home* do usuário.

Exemplo:

```
$ pwd
```

```
/var/spool
```

```
$ cd
```

```
$ pwd
```

```
/home/fabio
```

```
$ cd /etc
```

```
$ pwd
```

```
/etc
```

No caso acima, o diretório de trabalho era */var/spool*. Ao executar o comando “*cd*” o diretório de trabalho mudou para */home/fabio*. Ao executar o comando “*cd /etc*” o diretório de trabalho mudou para */etc*.

Algumas utilizações do comando *cd*:

**cd /** → Vai para o diretório raiz (root);

**cd ..** → Sobe um nível na árvore de diretórios;

**cd -** → Retorna para o diretório que estava antes de entrar no diretório atual.

## MKDIR

Cria um ou mais diretórios.

Sintaxe: ***mkdir [caminho/diretório]***

Na qual *caminho* é o local em que o diretório será criado e *diretório* é o nome do diretório a ser criado.

Algumas opções:

**-p** → Caso os diretórios dos níveis acima não existam, serão criados.

Exemplo:

**\$ mkdir a/b/c**

**mkdir:** não foi possível criar o diretório “a/b/c”: Arquivo ou diretório não encontrado

**\$ mkdir -p a/b/c**

**\$ mkdir a/b/c/d**

No exemplo, o primeiro comando (**mkdir a/b/c**) falhou porque tentou criar, a partir do diretório corrente, o diretório “c” dentro do caminho “a/b”, mas o caminho “a/b” não existe. O segundo comando (**mkdir -p a/b/c**) funcionou por causa do parâmetro **-p**, que forçou a criação dos diretórios “a” e “a/b” caso não existissem. O terceiro comando criou o diretório “d” dentro do caminho “a/b/c”.

## RMDIR

Remove um diretório vazio.

Sintaxe: ***rmdir [caminho/diretório]***

Exemplo:

**\$ rmdir a/b/c/d**

**\$ rmdir a**

**rmdir:** falhou em remover 'a': Diretório não vazio

Neste caso, o primeiro comando funcionou e removeu o diretório “d” que estava no caminho “a/b/c”. O segundo comando não funcionou porque tentou remover o diretório “a”, que não está vazio. Para a remoção de diretórios que contenham arquivos e/ou subdiretórios, consulte o comando *rm*.

# COMANDOS PARA MANIPULAÇÃO DE ARQUIVOS

## RM

O comando *rm* é utilizado para apagar arquivos. Pode ser utilizado também para apagar diretórios e subdiretórios recursivamente.

Sintaxe: ***rm [opções] arquivo/diretório***

Algumas opções:

-i → Pergunta antes de remover;

-v → Mostra os nomes dos arquivos/diretórios conforme são removidos;

-r → Remove arquivos e diretórios recursivamente;

-f → Remove sem perguntar.

Utilizando a opção “-r” é possível apagar diretórios recursivamente. Desta forma, sempre que for necessário apagar um diretório que contenha arquivos, o comando “rm -r” deve ser utilizado no lugar do comando “rmdir”.

Exemplo:

```
$ rm -rf a
```

Neste caso, o arquivo *a* é removido, pois sendo *a* um diretório, ele será apagado, assim como todos os seus arquivos e subdiretórios, sem que seja solicitada a confirmação.

## CP

Copia arquivos e diretórios. É necessário especificar tanto a origem quanto o destino do arquivo/diretório a ser copiado.

Sintaxe: ***cp [opções] origem destino***

Na qual *origem* indica o arquivo ou diretório a ser copiado e *destino* indica o local para onde deve ser copiado.

Algumas opções:

-i → Pergunta antes de substituir um arquivo existente;

**-f** → Substitui arquivos existentes sem perguntar;

**-r** → Copia arquivos dos diretórios e subdiretórios da origem para o destino;

**-R** → Copia arquivos e diretórios recursivamente, assim como os arquivos especiais FIFO e dispositivos;

**-v** → Mostra os nomes dos arquivos que estão sendo copiados;

**-p** → Preserva atributos do arquivo;

**-u** → Copia somente se o arquivo de origem é mais novo que o arquivo de destino ou quando o arquivo de destino não existe.

Exemplo:

```
$ cp /etc/fstab .
```

```
$ cp teste.txt outro.txt
```

O primeiro exemplo copia o arquivo *fstab* que está no diretório */etc* para o diretório de trabalho (“.”). O segundo exemplo copia o arquivo *teste.txt* para *outro.txt* no mesmo diretório.

## MV

Move ou renomeia arquivos e diretórios.

Sintaxe: ***mv [opções] origem destino***

Em que *origem* indica o arquivo ou diretório a ser movido e *destino* indica o local para onde deve ser movido.

Algumas opções:

**-f** → Substitui o destino sem perguntar;

**-i** → Pergunta antes de substituir;

**-v** → Mostra os nomes dos arquivos que estão sendo movidos;

**-u** → Move somente se o arquivo origem for mais novo que o arquivo destino, ou se o arquivo não existir no destino.

Exemplo:

```
$ mv teste.txt outro.txt
```

```
$ mv outro.txt a/b/
```

No exemplo, o primeiro comando renomeia o arquivo *teste.txt* para *outro.txt*. O segundo comando move o arquivo *outro.txt* para o diretório *b* que está dentro do diretório *a*.

## CAT

Mostra o conteúdo de um arquivo.

Sintaxe: **cat [opções] arquivo**

Algumas opções:

-n → Mostra o número das linhas enquanto o conteúdo do arquivo é exibido;

-s → Não mostra mais que uma linha em branco entre um parágrafo e outro.

Exemplo:

```
$ cat /etc/host.conf
```

```
# The "order" line is only used by old versions of the C library.
```

```
order hosts,bind
```

```
multi on
```

## FIND

Procura por arquivos/diretórios no disco, podendo ser por intermédio de diversas opções, como nome, data de modificação, tamanho etc.

Sintaxe: **find [diretório] [opções/expressão]**

Em que *diretório* é o local onde se inicia a busca, percorrendo todos os seus subdiretórios.

Algumas opções/expressão:

-name [expressão] → Procura pelo nome [expressão];

-amin [num] → Procura por arquivos que foram acessados [num] minutos atrás. Caso for antecedido por “-” (menos), procura por arquivos que foram acessados entre [num] minutos atrás até agora;

-atime [num] → Procura por arquivos que foram acessados [num] dias atrás. Caso for antecedido por “-” (menos), procura por arquivos que foram acessados entre [num] dias atrás e a data atual;

-gid [num] → Procura por arquivos que possuam a identificação numérica do grupo igual a [num];

-group [nome] → Procura por arquivos que possuam a identificação de nome do grupo igual a [nome];

-uid [num] → Procura por arquivos que possuam a identificação numérica do usuário igual a [num];

-user [nome] → Procura por arquivos que possuam a identificação de nome do usuário igual a [nome];

-mmin [num] → Procura por arquivos que tiveram seu conteúdo modificado há [num] minutos.

Caso for antecedido por “-” (menos), procura por arquivos que tiveram seu conteúdo modificado entre [num] minutos atrás até agora;

-mtime [num] → Procura por arquivos que tiveram seu conteúdo modificado há [num] dias. Caso for antecedido por “-” (menos), procura por arquivos que tiveram seu conteúdo modificado entre [num] dias atrás até agora;

-perm [modo] → Procura por arquivos que possuam os modos de permissão [modo];

-size [num] → Procura por arquivos que tiverem o tamanho [num]. [num] pode ser antecedido de “+” ou “-” para especificar um arquivo maior ou menor que [num].

Exemplo:

```
$ find /home -name outro.txt
```

```
/home/fabio/a/b/outro.txt
```

Neste caso, o comando procura por um arquivo ou diretório de nome *outro.txt*, fazendo a busca a partir do diretório */home*. O retorno do comando informa que o arquivo se encontra no diretório */home/fabio/a/b*.

## LINKS SIMBÓLICOS E HARDLINKS

**Links simbólicos** são entradas de diretório que, ao invés de indicar um local no disco onde se encontram dados de um arquivo, apontam para outras entradas de diretórios. Funcionam como atalho para um arquivo ou diretório.

Para efeito de utilização, não faz diferença acessar um arquivo diretamente ou por intermédio de um link simbólico. O processo é totalmente transparente para o usuário.

Tais entradas de diretório são bastante úteis para compatibilizar sistemas que procuram por arquivos em diferentes locais. No lugar de manter cópias do mesmo arquivo, cria-se links simbólicos nos diferentes locais onde os arquivos podem ser procurados, economizando espaço em disco e evitando inconsistências que poderiam ser provocadas por arquivos que deveriam ser iguais mas possuem conteúdo diferente.

No Linux, um arquivo é acessado por meio de seu i-node. Uma entrada de diretório de arquivo deve apontar para o i-node do arquivo de forma a poder acessar o seu conteúdo. O sistema de arquivos do Linux permite que diferentes entradas de diretórios apontem para o mesmo i-node, fazendo com que essas entradas apontem, na prática, para o mesmo arquivo.

## DICA

A maneira de fazer com que diferentes entradas apontem para o mesmo arquivo se dá através de um **hardlink** (denominado link duro por alguns autores).

Para criar links simbólicos e hardlinks, utiliza-se o comando **In**, cuja sintaxe é:

**In [opções] alvo [nome\_do\_link/diretório]**

**alvo** é o arquivo/diretório que será referenciado pelo link;

**nome\_do\_link/diretório** é o nome do link que será criado ou o diretório onde será criado o link com o mesmo nome do alvo.

Algumas opções:

**-s** → Cria um link simbólico;

**-v** → Mostra o nome de cada arquivo antes de fazer o link;

**-d** → Cria hardlink para diretórios.

Suponha que em um diretório exista um arquivo de nome “*documento.txt*”. O comando abaixo cria no mesmo diretório um link simbólico chamado “*simbolico.txt*” que aponta para “*documento.txt*”.

```
$ In -s documento.txt simbolico.txt
```

```
$ ls -lh
```

```
total 4,0K
```

```
-rw-r--r-- 1 fabio fabio 2,8K ago 29 21:05 documento.txt
```

```
lrwxrwxrwx 1 fabio fabio 13 ago 29 21:21 simbolico.txt -> documento.txt
```

Observe pela saída do comando “ls -lh” que “simbolico.txt” é um link simbólico para “documento.txt”.

Para criar um hardlink de nome “hard.txt” para o arquivo “documento.txt”, é utilizado também o comando **In**, mas sem a opção **-s**.

```
$ In documento.txt hard.txt
```

```
$ ls -lhi
```

```
total 8,0K
```

```
404662 -rw-r--r-- 2 fabio fabio 2,8K ago 29 21:05 documento.txt
```

```
404662 -rw-r--r-- 2 fabio fabio 2,8K ago 29 21:05 hard.txt
```

```
396432 lrwxrwxrwx 1 fabio fabio 13 ago 29 21:21 simbolico.txt -> documento.txt
```

A opção **-i** do comando **ls** (para listar o conteúdo do diretório) informa o número do i-node do arquivo.

Observe no exemplo acima que ambos “documento.txt” e “hard.txt” possuem o mesmo número de i-node (404662), ou seja, ambos apontam para o mesmo arquivo.

Qualquer alteração feita no arquivo sendo acessado por qualquer um dos 3 nomes (“documento.txt”, “simbolico.txt” ou “hard.txt”) acarretará alteração para todos, já que apontam para o mesmo arquivo.

Observe também, no último exemplo, para no número 2 antes do nome do dono do arquivo (fabio). Isso indica que existem duas entradas de diretório apontando para o mesmo i-node (404662). O link simbólico não é computado.

## RESUMINDO

Isso significa que o sistema de arquivos mantém uma contagem de quantas entradas de diretório estão apontando para o mesmo i-node (o mesmo arquivo).

Assim, se alguma das entradas de diretório for eliminada por **rm** ou por **rmdir**, o arquivo/diretório será efetivamente eliminado somente se existir uma única entrada de diretório apontada para ele. Se houver mais de um hardlink para o arquivo, somente a entrada de diretório é apagada.

# MANIPULANDO ARQUIVOS NO LINUX

Assista, a seguir, à explicação do professor sobre como manipular arquivos:

# VERIFICANDO O APRENDIZADO

## MÓDULO 4

---

- Descrever o funcionamento dos principais editores de arquivos do Linux

## EDITOR DE ARQUIVOS X PROCESSADOR DE TEXTOS

Arquivos de texto puro e documentos criados por processadores de texto como o Microsoft Word ou o LibreOffice Writer possuem diferenças significativas.

Um arquivo de texto puro contém caracteres **ASCII**, cujo conteúdo pode ser entendido visualizando diretamente o conteúdo do arquivo, sem necessidade de conversões ou interpretações.

## ASCII

ASCII (*American Standard Code for Information Interchange* - Código Padrão Americano para o Intercâmbio de Informação) é um padrão de codificação de caracteres criado para padronizar a forma como os computadores representam letras, números, acentos, sinais diversos e alguns códigos de controle.

## SAIBA MAIS

Alguns arquivos deste tipo, como o HTML, possuem marcações de controle, mas todos compostos por caracteres ASCII.

Já documentos criados por um processador de texto possuem conteúdo com formato específico, possuindo vários caracteres de controle e, comumente, informações binárias. Neste tipo de arquivo, é possível seleção de diferentes fontes para texto, com diferentes tamanhos e formatações (negrito, itálico etc.), além da incorporação de tabelas, imagens e links, entre outros.

Um processador de textos muito utilizado por usuários do Linux é o Writer, ferramenta integrante do pacote LibreOffice, um software livre de escritório com software para processamento de texto, planilha eletrônica, apresentação, desenho, banco de dados, edição de equações matemáticas etc.



Arquivos de texto puro são comumente utilizados para configuração de sistemas em um ambiente Linux, e um administrador de sistemas precisa saber como editar tais arquivos. Neste módulo, conheceremos alguns dos principais editores de texto puro disponíveis para utilização em sistemas Linux.

## VIM

Vim é uma abreviação de Vi Improved (Vi Melhorado), sendo uma melhoria do editor de textos **vi**, criado em 1976 para o **BSD**. O vi destacou-se por ser um editor pequeno e leve, podendo ser colocado junto de mídias com pouco espaço de armazenamento para ser executado durante operações de manutenção de emergência, ou em sistemas com pouco recurso computacional disponível. Por essas características, é um editor de textos disponível em toda distribuição Linux, que pode ser utilizado tanto em emergências quanto no dia a dia para alterar arquivos de configuração do sistema.

## BSD

Berkeley Software Distribution (BSD) é um sistema operacional Unix desenvolvido na Universidade da Califórnia em Berkeley.

O vim é um editor modal, ou seja, possui diferentes modo de operação (*comando* e *edição*) e as teclas possuem diferentes funções em cada modo. Ele é executado a partir de um terminal, sendo colocado em execução digitando o comando “*vim nome\_do\_arquivo*” ou “*vi nome\_do\_arquivo*”, no qual *nome\_do\_arquivo* é o nome do arquivo que será executado. A opção por utilizar vi ou vim depende da distribuição.

## ★ EXEMPLO

No Ubuntu 20.04 LTS, utilizado em nossos exemplos, deve ser aplicada a forma “*vi nome\_do\_arquivo*”.

Sempre que o editor é aberto, ele **inicia no modo de comando**. Nesse modo, quando se digita algo no teclado, são enviados comandos para que o editor execute determinadas ações sobre o texto que está sendo editado. Quando estiver no modo de edição, todo o texto digitado será inserido na posição onde se encontra o cursor.

## 💡 DICA

Para sair do modo de edição e voltar ao modo de comando, pressione a tecla **<Esc>**.

# PRINCIPAIS COMANDOS DO VIM

Para que você possa utilizar o vim, serão apresentados alguns comandos básicos para utilização deste poderoso editor de textos.

## ENTRAR EM MODO DE EDIÇÃO

i → Insere na posição atual.

I → Insere no começo da linha.

a → Acrescenta na posição atual.

A → Acrescenta ao final da linha.

o → Insere na linha abaixo, criando uma nova linha.

O → Insere na linha acima, criando uma nova linha.

cc → Deleta a linha e entra no modo de inserção.

C → Deleta linha a frente do cursor e entra em modo de inserção.

s → Deleta caractere e entra no modo de inserção.

S → Deleta linha e entra no modo de inserção.

## **SAIR DO MODO DE EDIÇÃO**

Pressionar a tecla <Esc>.

## **SALVAR E SAIR**

:w → Salva as alterações.

:w arquivo → Salva as alterações no arquivo especificado. Funciona como “save as”. Continuará editando (e salvando) o arquivo com o nome anterior.

:sav arquivo → Salva as alterações no arquivo especificado. Funciona como “save as”. Continuará editando (e salvando) o arquivo com o novo nome.

:q → Sai.

:q! → Força a saída sem salvar as alterações.

:wq → Salva as alterações e sai.

## **MOVIMENTAÇÃO**

A forma mais comum para movimentar o cursor ainda é por meio das setas do teclado, mas podem ser utilizadas também as teclas alfabéticas.

j → Baixo.

k → Cima.

l → Direita.

h → Esquerda.

0 → Volta ao começo da linha (tecla zero).

^ → Volta ao começo da linha (duas vezes já que se trata de um acento).

\$ → Vai até o final da linha.

w → Avança até a próxima palavra.

e → Avança até o fim da palavra atual.

**b** → Retorna ao início da palavra.

**f[caractere]** → Pressione f seguido de algum caractere para posicionar o cursor na próxima ocorrência desse caractere.

**t[caractere]** → A mesma coisa para o f, mas posiciona um caractere antes do caractere pressionado.

**gg** → Retorna à primeira linha.

**G** → Vai até a última linha.

**:[número da linha]** → Vai até a linha especificada.

**''** → Volta até onde você estava antes de pular de posição.

## **APAGANDO**

Em **modo de edição**, utilize as teclas <Backspace> e <Delete> para as correções normais. Em **modo de comando** podem ser realizadas exclusões mais elaboradas.

**x** → Apaga o caractere sob o cursor.

**X** → Apaga o caractere antes do cursor.

**D** → Apaga da posição atual até o fim da linha.

**J** → Junta duas linhas. 5J juntará cinco linhas contíguas.

**dd** → Apaga toda a linha.

**dj** → Apaga 2 linhas abaixo.

**dk** → Apaga 2 linhas acima.

**dw** → Apaga até o fim da palavra.

**dt"** → Apaga da posição atual até o fechamento das aspas.

**5db** → Apaga cinco palavras para trás.

**d[algum comando de posicionamento]** - Combina o comando com qualquer outro comando de posicionamento, veja os exemplos abaixo:

## **DESFAZENDO ALTERAÇÕES**

**u** → Desfazer.

**^r** → Refazer.

## **COPIANDO**

**yy** → Copia toda a linha.

**Y** → Copia toda a linha.

yw → Copia até o fim da palavra.

y2j → Copia mais duas linhas abaixo.

"+y → Copia para a área de transferência.

## COLANDO

p → Cola a partir da posição atual.

P → Cola na posição atual.

[p → Colar antes.

]p → Colar depois.

"+gp → Colar da área de transferência.

## REPETIÇÃO DE COMANDOS

Para repetir um comando, digite antes um número representando a quantidade de vezes que deseja repeti-lo. Exemplo:

3w - Avança três palavras

10k - Sobe dez linhas

2t" - Coloca o cursor antes da segunda aspas

3i[escreva e pressione a tecla <Esc>] → O que for digitado será inserido 3 vezes.

## BUSCAR

/txt → Busca pelo termo txt. Para ignorar a diferença entre maiúsculas e minúsculas, basta incluir \c no termo da busca: /\ctxt - Realiza uma busca do termo digitado sem diferenciar se está em maiúsculo ou minúsculo. /\Ctxt (C maiúsculo) força a diferenciação de maiúsculas e minúsculas.

n → Localiza a próxima ocorrência.

N → Localiza a ocorrência na direção contrária.

\* → Localiza palavra sob o cursor.

:set hlsearch → Destaca todos os termos encontrados (highlight).

:set nohlsearch → Desabilita a funcionalidade.

:set ignorecase → Configura todas as buscas para não diferenciar maiúsculas e minúsculas.

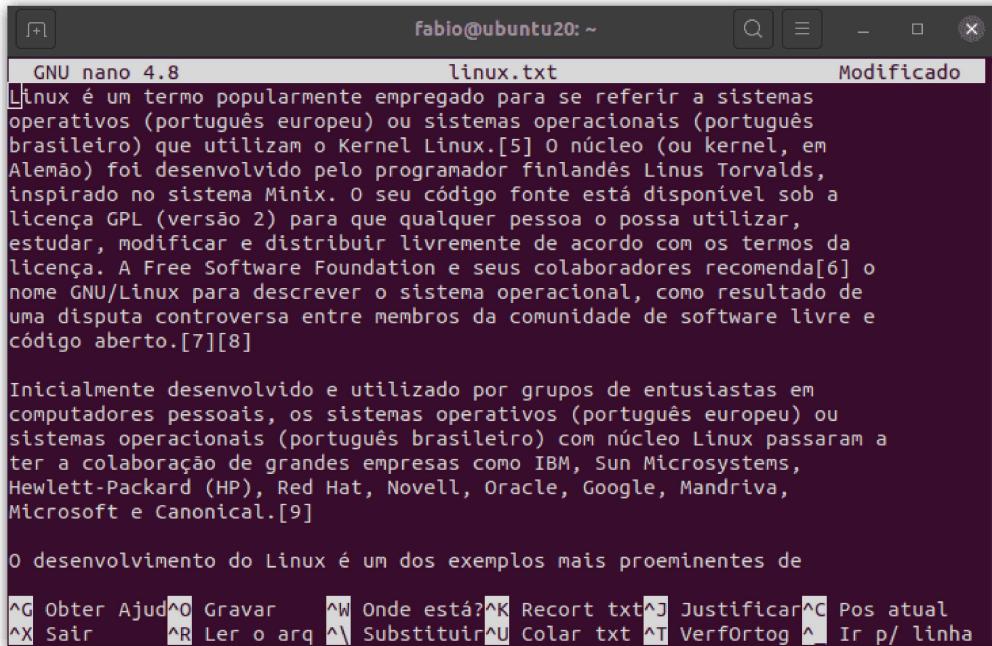
É possível utilizar expressões regulares como padrão de busca!

Clique **aqui** para fazer o download de um documento contendo alguns comandos e recursos mais avançados do vim.

# NANO

Assim como vim, o nano é também um editor de texto para linha de comando que está presente na grande maioria das distribuições Linux. Mas ao contrário do vim, é um editor que não possui diferentes modos de operação, sendo bem mais intuitiva sua utilização.

O nano é executado a partir de um terminal digitando o comando “*nano nome\_do\_arquivo*”, no qual *nome\_do\_arquivo* é o nome do arquivo que será executado. O programa já inicia sua execução com o arquivo aberto para execução.



A screenshot of a terminal window titled "fabio@ubuntu20: ~". The window shows the "GNU nano 4.8" editor with a file named "linux.txt" open. The text in the file discusses the history of Linux, mentioning Linus Torvalds, the GPL license, and the Free Software Foundation. At the bottom of the screen, a series of keyboard shortcuts are displayed:

```
^G Obter Ajuda ^O Gravar      ^W Onde está? ^K Recort txt ^J Justificar ^C Pos atual
^X Sair          ^R Ler o arq ^\ Substituir ^U Colar txt ^T VerfOrtog ^_ Ir p/ linha
```

Editor de texto nano executando em um terminal do Linux.

Os principais comandos do nano ficam disponíveis de forma visual na parte inferior da tela. São eles:

<CTRL>+G → Obter ajuda;

<CTRL>+O → Salvar o arquivo;

<CTRL>+X → Sair;

<CTRL>+R → Abrir arquivo;

<CTRL>+W → Procurar no texto;

<CTRL>+\ → Pesquisar e substituir;

<CTRL>+K → Copia a linha para a memória;

<CTRL>+U → Cola o que estiver na memória;

<CTRL>+J → Quebra a linha em várias linhas de forma a caberem na janela;

<CTRL>+C → Mostra uma linha indicando a posição em que se encontra no texto;

<CTRL>+L → Ir para a linha.

## DICA

Consulte a lista de todos os comandos disponíveis abrindo o editor de textos nano e teclando <CTRL>+G.

# GEDIT

É bom que o administrador de sistemas Linux tenha conhecimentos de utilização de editores de texto de linhas de comando, pois em uma emergência, em que não é possível utilizar o sistema pela interface gráfica, ele poderá editar os arquivos de configuração do sistema sem maiores dificuldades. Porém, quando existe uma interface gráfica disponível para utilização, pode ser mais confortável a utilização de um editor de textos em modo gráfico.

Como as distribuições Linux são livres para escolherem as interfaces gráficas que comporão o sistema, e cada uma possui seu próprio editor de textos puro, o editor utilizado na interface gráfica dependerá da interface utilizada.

O Ubuntu 20.04 LTS, sistema operacional Linux aplicado em nossos exemplos, utiliza o GNOME como interface gráfica. O editor de textos puro do GNOME é conhecido como **gedit** e pode ser colocado em execução de diferentes formas.

A primeira delas é navegando pelos arquivos utilizando o gerenciador de arquivos do GNOME e clicando duas vezes sobre o nome do arquivo. Se for um arquivo cujo formato esteja configurado para ser aberto no gedit, ele será colocado em execução.

A segunda forma é abrindo a tela de aplicativos do GNOME e selecionando o gedit. Normalmente aparece com o nome “*Editor de texto*”.

Uma terceira forma de abrir o gedit é por intermédio do *shell*. Abra um terminal, entre no diretório onde se encontra o arquivo que deseja editar e entre com o comando “**gedit <nome do arquivo>**” para que o arquivo seja aberto no gedit. Ou simplesmente digite gedit para iniciar o gedit.

Na imagem a seguir podemos ver a tela do gedit com um arquivo em edição:



📷 Tela do gedit (editor padrão do GNOME).

No canto superior esquerdo da tela (1) existe um botão “Abrir”. Ele pode ser utilizado para abrir um novo documento no gedit.

## 📣 ATENÇÃO

Ao abrir um novo documento, o anterior também permanece em edição, sendo que a tela do gedit é dividida verticalmente para permitir a visualização de ambos.

À direita do botão, existe um triângulo invertido (2) que pode ser utilizado para buscar rapidamente os últimos arquivos que foram abertos no editor. Ainda próximo ao triângulo, existe um botão (3) utilizado quando se deseja criar um arquivo.

No centro (4) aparece o nome do arquivo e edição, e o nome do diretório onde o arquivo se encontra. O botão “Salvar” (5) é utilizado para salvar o arquivo em edição a qualquer momento.

À direita do botão “Salvar” (6) fica o botão para acesso ao menu do editor, onde existem várias opções e ferramentas, entre elas opções para localizar e imprimir. Abaixo na janela, na barra de status, é possível selecionar o tipo do arquivo em edição (7).

## DICA

Normalmente, o gedit seleciona automaticamente o tipo do arquivo e utiliza-o para auxiliar o usuário na edição do arquivo.

No exemplo, está sendo editado um arquivo em Linguagem C, então o gedit auxilia fazendo automaticamente as tabulações e colorindo as palavras reservadas por tipo. É possível também escolher a largura da tabulação (8) adicionada quando é pressionada a tecla “Tab”. À direita, é informada a posição do cursor (linha, coluna) e por fim é mostrado (10) se a entrada de dados está em modo de *inserção* ou de *sobreposição*.

## EDITORES DE TEXTO NO LINUX

Veja, a seguir, como usar editores de texto:



## VERIFICANDO O APRENDIZADO

## CONCLUSÃO

## CONSIDERAÇÕES FINAIS

Acabamos de estudar o funcionamento dos sistemas de arquivos utilizados pelos sistemas operacionais modernos. Para isso, iniciamos nossa jornada entendendo os conceitos de arquivos e diretórios, e como são as técnicas de implementação destes objetos pelos diferentes sistemas de arquivos. Além disso, vimos como um sistema de cache é importante para aumentar o desempenho de acesso.

Após o estudo conceitual, observamos como administrar um sistema de arquivos no sistema operacional Linux. Para tanto, compreendemos como é o funcionamento de um disco rígido e como particioná-lo.

Após particionar e definir o tipo do sistema de arquivos, aprendemos a formatá-lo e fazer sua montagem sobre o sistema de arquivos raiz do sistema operacional.

No entanto, não basta criar e montar o sistema de arquivos, é preciso também saber como gerenciá-lo.

Por conta disso, aprendemos diversos comando para gerenciamento de arquivos e diretórios no Linux, assim como criar links simbólicos e hardlinks.

Por fim, conhecemos sobre os principais editores de texto puro utilizados pelo Linux. Uma percepção importante para que os administradores de sistemas Linux possam recuperar o sistema na eventualidade de uma falha grave.



PODCAST

## ⚠ PODCAST

# REFERÊNCIAS

ARCHLINUX. **Partitioning**. Consultado em meio eletrônico em 22 ago. 2020.

DEITEL, H.M. **Sistemas Operacionais**. 3 ed. São Paulo: Pearson Prentice Hall, 2005.

MACHADO, F. B. & MAIA, L. P. **Arquitetura de Sistemas Operacionais**. 5 ed. Rio de Janeiro: LTC, 2013.

NEMETH, E. et al. **Manual Completo do Linux: Guia do Administrador.** 2 ed. São Paulo: Pearson Prentice Hall, 2007.

UBUNTU. **Official Ubuntu Documentation.** Consultado em meio eletrônico em 2 ago. 2020.

SILBERSCHATZ, A. **Fundamentos de Sistemas Operacionais.** 9 ed. Rio de Janeiro: LTC, 2013.

SILBERSCHATZ, A. & GALVIN, P. B. **Sistemas Operacionais – Conceitos.** 5 ed. São Paulo: Prentice Hall, 2000.

SILVA, G. M. **Guia Foca GNU/Linux.** In: Guia Foca.

SOARES, L. F. G. **Redes de Computadores – Das LANs, MANs e WANS às Redes ATM.** 2 ed. Rio de Janeiro: Campus, 1995.

TANENBAUM, A. S. & BOS, H. **Sistemas Operacionais Modernos.** 4 ed. São Paulo: Pearson Educacional do Brasil, 2016.

TANENBAUM, A. S. & WOODHULL, A. S. **Sistemas Operacionais – Projeto e Implementação.** 3 ed. Porto Alegre: Bookman, 2008.

TANENBAUM, A. S. **Redes de Computadores.** 5 ed. São Paulo: Pearson, 2011.

TANENBAUM, A. S. **Structured Computer Organization.** 3 ed. Prentice Hall International, 1990.

Página consultada: **NBCGIB**

---

## EXPLORE+

Para saber mais sobre os assuntos tratados neste tema, pesquise:

**LVM (Logical Volume Manager** — gerenciador de volume lógico) disponível em Archlinux;

Tópicos: **Listas de controle de acesso e Adicionando novos usuários** do livro NEMETH, E. et al. **Manual Completo do Linux: Guia do Administrador.** 2 ed. São Paulo: Pearson Prentice Hall, 2007.

Acesse os sites:

**Distrowatch.** Sobre distribuições Linux;

**Libreoffice.** Para mais detalhes sobre essa ferramenta.

---

# CONTEUDISTA

Fábio Contarini Carneiro

 CURRÍCULO LATTES