Introdução ao Linux

Arthur Mendes Peixoto
Andreia Gentil Bonfante

A RNP – Rede Nacional de Ensino e Pesquisa – é qualificada como uma Organização Social (OS), sendo ligada ao Ministério da Ciência, Tecnologia e Inovação (MCTI) e responsável pelo Programa Interministerial RNP, que conta com a participação dos ministérios da Educação (MEC), da Saúde (MS) e da Cultura (MinC). Pioneira no acesso à Internet no Brasil, a RNP planeja e mantém a rede Ipê, a rede óptica nacional acadêmica de alto desempenho. Com Pontos de Presença nas 27 unidades da federação, a rede tem mais de 800 instituições conectadas. São aproximadamente 3,5 milhões de usuários usufruindo de uma infraestrutura de redes avançadas para comunicação, computação e experimentação, que contribui para a integração entre o sistema de Ciência e Tecnologia, Educação Superior, Saúde e Cultura.

RNP

Ministério da **Cultura**

Ministério da **Saúde**

Ministério da **Educação**

Ministério da **Ciência, Tecnologia e Inovação**

Introdução **ao Linux**

Arthur Peixoto
Andreia Gentil Bonfante

Introdução **ao Linux**

Arthur Peixoto
Andreia Gentil Bonfante

Rio de Janeiro
Escola Superior de Redes
2013

Rua Lauro Müller, 116 sala 1103
22290-906 Rio de Janeiro, RJ

Diretor Geral
Nelson Simões

Diretor de Serviços e Soluções
José Luiz Ribeiro Filho

Escola Superior de Redes

Coordenação
Luiz Coelho

Edição
Pedro Sangirardi

Revisão Técnica
Marcelo Castellan Braga

Coordenação Acadêmica de Administração de Sistemas
Sergio Alves de Souza

Equipe ESR (em ordem alfabética)
Celia Maciel, Cristiane Oliveira, Derlinéa Miranda, Edson Kowask, Elimária Barbosa, Lourdes Soncin, Luciana Batista, Luiz Carlos Lobato e Renato Duarte

Capa, projeto visual e diagramação
Tecnodesign

Versão
1.1.0

Distribuição
Escola Superior de Redes
Rua Lauro Müller, 116 – sala 1103
22290-906 Rio de Janeiro, RJ
http://esr.rnp.br
info@esr.rnp.br

Dados Internacionais de Catalogação na Publicação (CIP)

P377a Peixoto, Arthur Mendes
Introdução ao Linux / Arthur Mendes Peixoto, Andreia Bonfante; Revisor: Marcelo Braga. – Rio de Janeiro: RNP/ESR, 2013.
210 p. : il. ; 28 cm.

Bibliografia: p. 193-194.
ISBN 978-85-63630-19-3

1. Linux (Sistema operacional de computador). 2. Sistema operacional (computadores). 3. UNIX Shell (Programa de computador). I. Bonfante, Andreia. II. Braga, Marcelo. III. Titulo.

CDD 005.40469

Sumário

3. Organização do Linux

Escola Superior de Redes

A Escola Superior de Redes (ESR) é a unidade da Rede Nacional de Ensino e Pesquisa (RNP) responsável pela disseminação do conhecimento em Tecnologias da Informação e Comunicação (TIC).

A ESR nasce com a proposta de ser a formadora e disseminadora de competências em TIC para o corpo técnico-administrativo das universidades federais, escolas técnicas e unidades federais de pesquisa. Sua missão fundamental é realizar a capacitação técnica do corpo funcional das organizações usuárias da RNP, para o exercício de competências aplicáveis ao uso eficaz e eficiente das TIC.

A ESR oferece dezenas de cursos distribuídos nas áreas temáticas: Administração e Projeto de Redes, Administração de Sistemas, Segurança, Mídias de Suporte à Colaboração Digital e Governança de TI.

A ESR também participa de diversos projetos de interesse público, como a elaboração e execução de planos de capacitação para formação de multiplicadores para projetos educacionais como: formação no uso da conferência web para a Universidade Aberta do Brasil (UAB), formação do suporte técnico de laboratórios do Proinfo e criação de um conjunto de cartilhas sobre redes sem fio para o programa Um Computador por Aluno (UCA).

A metodologia da ESR

A filosofia pedagógica e a metodologia que orientam os cursos da ESR são baseadas na aprendizagem como construção do conhecimento por meio da resolução de problemas típicos da realidade do profissional em formação. Os resultados obtidos nos cursos de natureza teórico-prática são otimizados, pois o instrutor, auxiliado pelo material didático, atua não apenas como expositor de conceitos e informações, mas principalmente como orientador do aluno na execução de atividades contextualizadas nas situações do cotidiano profissional.

A aprendizagem é entendida como a resposta do aluno ao desafio de situações-problema semelhantes às encontradas na prática profissional, que são superadas por meio de análise, síntese, julgamento, pensamento crítico e construção de hipóteses para a resolução do problema, em abordagem orientada ao desenvolvimento de competências.

Dessa forma, o instrutor tem participação ativa e dialógica como orientador do aluno para as atividades em laboratório. Até mesmo a apresentação da teoria no início da sessão de aprendizagem não é considerada uma simples exposição de conceitos e informações. O instrutor busca incentivar a participação dos alunos continuamente.

As sessões de aprendizagem onde se dão a apresentação dos conteúdos e a realização das atividades práticas têm formato presencial e essencialmente prático, utilizando técnicas de estudo dirigido individual, trabalho em equipe e práticas orientadas para o contexto de atuação do futuro especialista que se pretende formar.

As sessões de aprendizagem desenvolvem-se em três etapas, com predominância de tempo para as atividades práticas, conforme descrição a seguir:

Primeira etapa: apresentação da teoria e esclarecimento de dúvidas (de 60 a 90 minutos). O instrutor apresenta, de maneira sintética, os conceitos teóricos correspondentes ao tema da sessão de aprendizagem, com auxílio de slides em formato PowerPoint. O instrutor levanta questões sobre o conteúdo dos slides em vez de apenas apresentá-los, convidando a turma à reflexão e participação. Isso evita que as apresentações sejam monótonas e que o aluno se coloque em posição de passividade, o que reduziria a aprendizagem.

Segunda etapa: atividades práticas de aprendizagem (de 120 a 150 minutos). Esta etapa é a essência dos cursos da ESR. A maioria das atividades dos cursos é assíncrona e realizada em duplas de alunos, que acompanham o ritmo do roteiro de atividades proposto no livro de apoio. Instrutor e monitor circulam entre as duplas para solucionar dúvidas e oferecer explicações complementares.

Terceira etapa: discussão das atividades realizadas (30 minutos). O instrutor comenta cada atividade, apresentando uma das soluções possíveis para resolvê-la, devendo ater-se àquelas que geram maior dificuldade e polêmica. Os alunos são convidados a comentar as soluções encontradas e o instrutor retoma tópicos que tenham gerado dúvidas, estimulando a participação dos alunos. O instrutor sempre estimula os alunos a encontrarem soluções alternativas às sugeridas por ele e pelos colegas e, caso existam, a comentá-las.

Sobre o curso

Esse é o curso introdutório da trilha de Administração de Sistemas. Seu objetivo é introduzir o aluno ao mundo Linux. O sistema operacional (SO) utilizado é o CentOS, que é baseado em RedHat, gratuito e extremamente estável, além de suportar todos os serviços necessários a um ambiente web.

O curso é composto de 10 capítulos de embasamento teórico e atividades correlatas para aprendizado e fixação do conhecimento. O curso tem como objetivo apresentar as facilidades de administração e gerenciamento, que serão exploradas com maior profundidade nos demais cursos da área de Administração de Sistemas da Escola Superior de Redes da RNP.

A quem se destina

Este curso é destinado a usuários, especialistas de suporte e desenvolvedores de software que desejam aprender a utilizar o Linux, um ambiente computacional moderno, ágil e com um sistema operacional extremamente estável e versátil.

O curso destina-se também aos administradores de sistemas Windows e aos profissionais que desejam iniciar os estudos para a certificação LPIC1, do Linux Professional Institute.

Convenções utilizadas neste livro

As seguintes convenções tipográficas são usadas neste livro:

Itálico
Indica nomes de arquivos e referências bibliográficas relacionadas ao longo do texto.

```
Largura constante
```

Indica comandos e suas opções, variáveis e atributos, conteúdo de arquivos e resultado da saída de comandos. Comandos que serão digitados pelo usuário são grifados em negrito e possuem o prefixo do ambiente em uso (no Linux é normalmente # ou $, enquanto no Windows é C:\).

Conteúdo de slide 
Indica o conteúdo dos slides referentes ao curso apresentados em sala de aula.

Símbolo 
Indica referência complementar disponível em site ou página na internet.

Símbolo 
Indica um documento como referência complementar.

Símbolo 
Indica um vídeo como referência complementar.

Símbolo 
Indica um arquivo de aúdio como referência complementar.

Símbolo 
Indica um aviso ou precaução a ser considerada.

Símbolo 
Indica questionamentos que estimulam a reflexão ou apresenta conteúdo de apoio ao entendimento do tema em questão.

Símbolo 
Indica notas e informações complementares como dicas, sugestões de leitura adicional ou mesmo uma observação.

Permissões de uso

Comentários e perguntas

Para enviar comentários e perguntas sobre esta publicação:
Escola Superior de Redes RNP
Endereço: Av. Lauro Müller 116 sala 1103 – Botafogo
Rio de Janeiro – RJ – 22290-906
E-mail: info@esr.rnp.br

Sobre os autores

Arthur Mendes Peixoto possui mais de 26 anos de experiência na área de Redes de Comunicação de Dados e Engenharia de Sistemas, com Dissertação de Mestrado em "Análise de Performance de Sistemas Distribuídos", no Instituto Militar de Engenharia - IME. Participou do desenvolvimento e implantação das primeiras redes com tecnologias ATM, Frame Relay, IP/ MPLS. Foi consultor de grandes projetos como o Backbone IP do Plano Nacional de Banda Larga (PNBL) da Telebrás. Trabalhou por 22 anos no setor de Telecomunicações da Embratel, atuando na prospecção de novas tecnologias para as Redes de Nova Geração – NGN. Participou de testes e verificações de requisitos de RFPs, nos países: EUA, Canadá, Japão, França, Espanha e México. Atuou no desenvolvimento de sistemas no CPqD - Campinas (SP).

Andreia Gentil Bonfante possui graduação em Bacharelado em Ciências de Computação pela Universidade Estadual de Londrina, mestrado e doutorado em Ciências da Computação e Matemática Computacional pela Universidade de São Paulo. Atualmente é professora/pesquisadora da Universidade Federal de Mato Grosso. Tem experiência na área de Ciência da Computação, com ênfase em Inteligência Artificial, atuando principalmente nos seguintes temas: Processamento de Língua Natural, Mineração de Textos e Aprendizado de Máquina. Atua na Educação a Distância como Coordenadora da Especialização em Informática na Educação. Atuou também como instrutora dos cursos de Introdução ao Linux da Escola Superior de Redes na Unidade de Cuiabá.

Marcelo Castellan Braga possui graduação em Engenharia Eletrônica pelo CEFET-RJ, pós-graduação em Análise, Projeto e Gerência de Sistemas pela PUC-RJ e mestrado em informática pela UNIRIO. Atualmente é sócio diretor da MCB Tech, empresa que presta consultoria em redes de computadores, serviços de internet, segurança de dados e desenvolvimento de software. Atuou durante mais de 10 anos na área de TI em empresas como Rede Nacional de Ensino e Pesquisa (RNP) e Embratel.

Sergio Ricardo Alves de Souza possui mais de 35 anos de experiência na área de Administração e Suporte de Sistemas. Trabalhou em empresas como: Burroughs (UNISYS), ARSA (Infraero), Cobra Computadores, LNCC e outras. Consultoria e treinamento em empresas como: Fiocruz, Jardim Botânico, Museu Goeldi, Cefet-MG, IBM Brasil. Participou do desenvolvimento e implantação de cursos profissionalizantes em Informática em Petrópolis - FAETC. Participou do projeto de criação do Instituto Superior de Tecnologia em Ciência da Computação de Petrópolis. Possui "Notório Saber em Informática" pelo LNCC.

1

Histórico e instalação

objetivos

Compreender o que é um Sistema Operacional e estudar a arquitetura do sistema operacional Unix.

conceitos

Histórico do Unix, hardwares suportados e instalação do Linux.

O que é um Sistema Operacional?

Funções básicas de um sistema operacional:

- Gerenciar o uso da CPU.
- Gerenciar o uso da memória RAM.
- Gerenciar o armazenamento de dados.
- Gerenciar os dispositivos de entrada e saída.
- Interpretar comandos.

Um Sistema Operacional é a interface de comunicação entre o usuário e o hardware. Para desempenhar essa função, o Sistema Operacional deve conhecer a linguagem do usuário e a do hardware, e também controlar a troca de mensagens entre os dois. Além disso, deve interpretar as ordens do usuário e passar ao hardware as instruções para que sejam executadas. Principais funções desempenhadas por um Sistema Operacional:

- **Gerenciar o uso da CPU**: o Sistema Operacional deve controlar a utilização da CPU, dividindo seu tempo de uso de modo que ela execute os processos dos usuários e do próprio Sistema Operacional, um de cada vez, enquanto os outros aguardam na fila para serem processados.
- **Gerenciar o uso da memória**: é preciso manter a integridade dos dados e dos programas em execução na memória RAM do computador.
- **Gerenciar os dispositivos de entrada e saída**: é função do Sistema Operacional gerenciar os acessos aos dispositivos de entrada e saída de dados (I/O), como: impressoras, monitores, teclados, scanners, microfones, caixas de som etc. Um subsistema, ou subconjunto de programas, realiza essa função específica do sistema, que consiste entre outras coisas na leitura e escrita de dados nesses periféricos.
- **Gerenciar o armazenamento de dados**: é função do Sistema Operacional armazenar e recuperar os dados nos dispositivos de armazenamento, como discos rígidos, pen drives, CDs, DVDs etc.

- **Interpretar comandos**: é necessário que o sistema interaja com o usuário. O interpretador de comandos recebe os pedidos ou os comandos e compreende o que o usuário deseja executar. Após interpretar o comando, encaminha pedidos aos outros módulos do Sistema Operacional, especializados em atender esses pedidos.

Estas funções básicas são as mínimas necessárias para o funcionamento de um computador. Veremos que um sistema operacional pode ter muitas outras funções que definem sua especialização.

Dependendo do tipo de aplicação que um computador vai suportar, seu Sistema Operacional pode necessitar de executar funções especiais, além das básicas apresentadas anteriormente. Um Sistema Operacional que se destina a aplicações pessoais deve ter interface gráfica bem desenvolvida para facilitar a utilização por usuários leigos em sistemas. Há aqueles que se destinam a aplicações em tempo real, ou seja, que são sensíveis a retardos, conforme ocorre com alguns controles de processos industriais. Existem computadores que são compartilhados, atendendo a usuários de uma empresa ou departamento, que necessitam realizar diversas tarefas simultâneas (multitarefa), ou que permitem vários usuários conectados simultaneamente (multiusuário). Esses computadores necessitam de Sistemas Operacionais que realizem funções especiais, como:

- **Controle de acesso**: administra o acesso de múltiplos usuários às informações armazenadas e aos dados em memória, garantindo a confidencialidade dessas informações. É função do Sistema Operacional garantir o sigilo às informações de cada usuário, restringindo o acesso a essas informações a outros usuários que utilizem o mesmo sistema.
- **Gerência de contabilização**: contabiliza todas as atividades do sistema, armazenando em disco as informações relativas às estatísticas de utilização, para posterior emissão de relatórios.

Veremos que, desde o nascimento do Unix até os dias atuais, além das funcionalidades básicas e especiais, várias outras funções foram agregadas ao Sistema Operacional, tornando-o um sistema de grande complexidade e aplicabilidade, ou seja, um Sistema Operacional multi-purpose.

Arquitetura do sistema operacional Unix

- Independência do hardware.
- Kernel faz a interface entre o hardware e o restante do sistema.
- Gerenciador de processos faz parte do kernel.

O Unix foi projetado com arquitetura em camadas, o que permite maior independência do hardware utilizado. Para que isso seja possível, apenas uma pequena parte do sistema possui acesso direto ao hardware e se comunica com o restante do sistema.

Somente o kernel depende do hardware, permitindo que quase todo o sistema seja reaproveitado na migração entre diferentes máquinas. O interpretador de comandos, chamado de shell, não faz parte do kernel do sistema, e é uma excelente linguagem de programação, que torna o Unix um sistema bastante versátil.

O gerenciador de processos faz parte do kernel do Sistema Operacional e é o responsável pelo gerenciamento dos processos em execução e pela divisão do tempo de processamento da CPU entre esses processos. Cada processo necessita de um período de tempo reservado para que possa ser executado pela CPU. Depois que esse tempo se esgota, outro processo passa a ser processado e o anterior passa a aguardar por um novo período de tempo para ser novamente processado. A cada processo é atribuída uma prioridade, que se altera

dinamicamente conforme a execução de um algoritmo no kernel do Unix. O gerenciador de processos também é responsável por gerenciar uma área em disco que se constitui numa extensão da memória RAM principal da máquina e que é utilizada em casos de esgotamento dessa memória. A área é chamada de área de swap, para onde são copiadas temporariamente imagens dos processos, liberando, assim, parte de memória para permitir que outros processos possam ser executados. A Figura 1.1 mostra como é dividida a arquitetura do Sistema Operacional Unix. Essa divisão proporciona a esse sistema diversas características, que veremos a seguir.

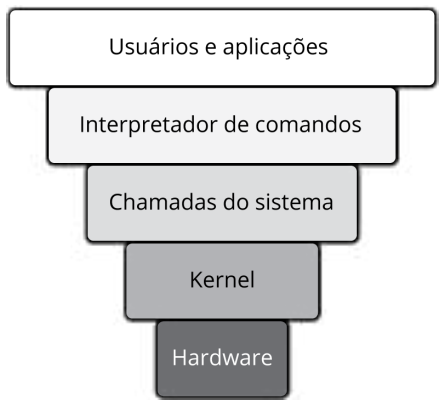


Figura 1.1
Divisão do Sistema Operacional Unix.

Características principais

- Portabilidade.
- Multiusuário.
- Multiprocessamento.
- Estrutura hierárquica de diretórios.
- Interpretador de comandos (shell).
- Pipelines.
- Utilitários.
- Desenvolvimento de software.
- Maturidade.

O Unix é um Sistema Operacional muito flexível, com grande número de funcionalidades. Principais características:

- **Portabilidade**: o Unix é portável, ou seja, pode ser adaptado facilmente para ser executado em diferentes arquiteturas de hardware. Sua adequação a um novo hardware é rápida e exige pequeno esforço de programação. Talvez essa seja a característica mais importante desse Sistema Operacional, que permitiu sua adoção por centenas de fabricantes diferentes. A portabilidade se estende, também, para os programas e pacotes de software escritos para o Unix, o que promoveu grande desenvolvimento de aplicativos e intensificou sua expansão no mercado.
- **Multiusuário**: o Unix foi concebido para ser um sistema multiusuário, suportando conexões simultâneas de diversos usuários. Com isso, é possível melhor utilização da capacidade de processamento e da manipulação e armazenamento das informações do sistema de computação. Para isso, o Sistema Operacional possui ferramentas de segurança para permitir o isolamento das atividades de cada usuário.

- **Multiprocessamento**: a funcionalidade de multiprocessamento do Unix permite a um usuário executar múltiplas tarefas simultaneamente. O sistema pode acessar um arquivo ou imprimir um relatório ao mesmo tempo em que o usuário pode editar um documento, possibilitando melhor produtividade e reduzindo a ociosidade tanto do processador quanto dos recursos de entrada e saída do sistema. A Figura 1.2 ilustra o conceito de multiprocessamento.

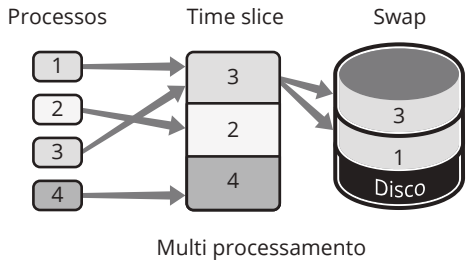


Figura 1.2
Multiprocessamento: execução de múltiplas tarefas ao mesmo tempo.

- **Estrutura hierárquica de diretórios**: o sistema de armazenamento de informações possui estrutura hierárquica, como mostra a Figura 1.3. A estrutura hierárquica é uma forma de arquivamento natural, pois pode ser comparada, por exemplo, à estrutura organizacional hierárquica de uma empresa e, consequentemente, torna mais fácil a localização e a manipulação de informações distribuídas por essa estrutura.

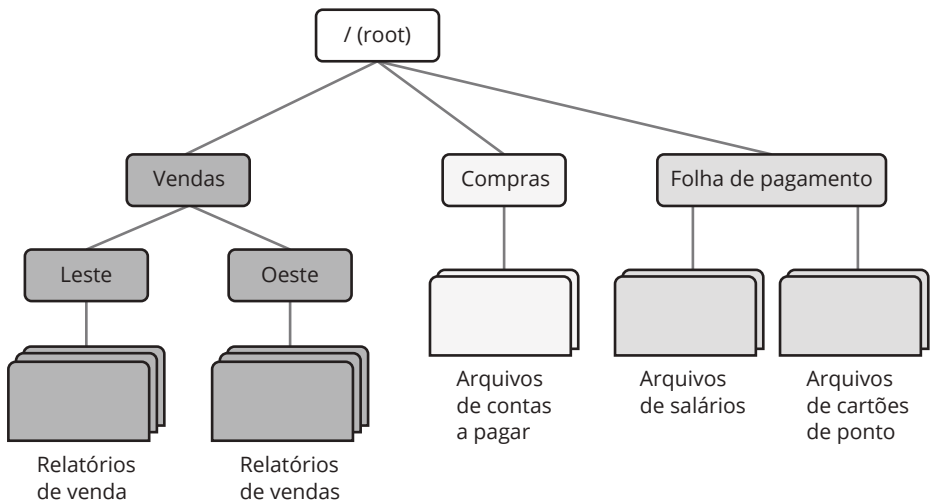


Figura 1.3
Estrutura de hierarquia do sistema de armazenamento de informações.

- **Interpretador de comandos (shell)**: a interação do usuário com o Sistema Operacional Unix é controlada por um poderoso interpretador de comandos, conhecido como shell. Esse interpretador suporta várias funcionalidades, como o redirecionamento de entrada e saída, a manipulação de grupos de arquivos com apenas um comando, a execução de sequências de comandos predefinidos, entre outras, facilitando a execução de tarefas complexas.
- **Pipelines**: o uso de pipelines ou pipes permite conectar a saída de um comando com a entrada de outro, como mostra a Figura 1.4. Essa é uma das mais famosas características do Unix e é utilizada para a execução de funções mais complexas. Muitas vezes, novas tarefas exigem apenas que programas já existentes e também utilitários sejam combinados para a execução de uma nova função, sem a necessidade de desenvolver um novo programa.

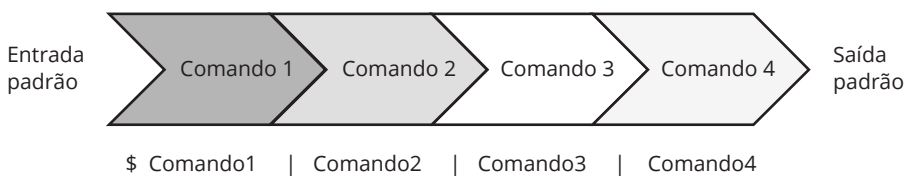


Figura 1.4
Uso de pipelines ou pipes.

- **Utilitários**: o Unix incorpora centenas de programas utilitários para funções como: seleção de dados, processamento de texto e busca de informações. Essas facilidades formam um conjunto de ferramentas que permitem a execução de diversos tipos de tarefas.
- **Desenvolvimento de software**: o Unix também é conhecido como uma plataforma de desenvolvimento de software, pois possui ferramentas que suportam todas as fases do processo de desenvolvimento, desde a preparação até a depuração. Devido à sua portabilidade, esses softwares podem ser utilizados em microcomputadores ou em computadores de grande porte.
- **Maturidade**: o Unix é um Sistema Operacional sólido, testado e aprovado pelo mercado, que vem sendo utilizado há mais de três décadas, tendo atingido o estado de maturidade. Dessa forma, além de sua flexibilidade e inúmeras funcionalidades, tem também como característica a confiabilidade, o que o torna, atualmente, o Sistema Operacional preferido para sistemas que suportam aplicações críticas.

Histórico do Unix

O Unix surgiu no Bell Laboratories em 1969, a partir do trabalho de Ken Thompson na evolução de outro Sistema Operacional, o Multics. No desenvolvimento do Unix, Thompson sentiu a necessidade de escrevê-lo para diferentes tipos de arquiteturas e decidiu fazer o Unix independentemente da máquina em que fosse executado, criando a linguagem B. Essa linguagem evoluiu para a linguagem C, desenvolvida por Dennis Ritchie, o que permitiu reescrever o Unix e torná-lo portável para diferentes arquiteturas. Devido à característica de portabilidade do Unix, no início da década de 1980 foram desenvolvidos diversos pacotes de programas, que foram utilizados por diferentes fabricantes, tornando mundialmente conhecidos o Unix, a linguagem C e seus aplicativos.

O Unix se tornou um Sistema Operacional bastante conhecido, não somente no mundo acadêmico, como também no meio comercial. Os sistemas Unix-based e similares ao Unix proliferaram, sendo oferecidos por diversos fabricantes de computadores, desde os pessoais até os mainframes, sempre mantendo suas características básicas, tornando-o um padrão de fato.

Versões do Unix

- Unix Sexta Edição.
- PWB Unix.
- Unix Versão 7.
- Unix System III.
- Unix System V.
- Berkeley Unix (BSD).

As versões do Unix são aquelas licenciadas exclusivamente pela AT&T, que podem ostentar o nome Unix. Uma variedade de versões, algumas mais famosas e outras menos foram desenvolvidas desde seu surgimento. Principais versões do Unix:

- **Unix Sexta Edição**: é a mais antiga versão licenciada, na maioria dos casos para instituições educacionais, não havendo nenhuma versão comercial dessa versão. Essa versão foi lançada em 1975.
- **PWB Unix**: versão especializada desenvolvida pelo Bell Labs, que inclui facilidades para desenvolvimento de software por grandes equipes de programadores. Essa versão foi lançada em 1977.

- **Berkeley Unix**: grande parte do desenvolvimento do Unix se deu na Universidade de Berkeley, na Califórnia, que se tornou um centro de atividades nos anos 70, externo ao Bell Labs/AT&T. As versões 4.1 BSD ou 4.2 BSD tinham seu próprio conjunto de utilitários e incorporavam suporte para superminicomputadores VAX, o editor de textos vi, e um shell especialmente adaptado para programação em C, chamado C Shell (csh). Esse sistema foi muito utilizado não somente nas universidades como também nas aplicações científicas e de engenharia. O System V, da AT&T, incorporou muitos desaes melhoramentos. Essa versão foi lançada em 1977.
- **Unix Versão 7**: primeira versão licenciada comercialmente pela AT&T, com maior número de instalações em microcomputadores. Essa versão foi lançada em 1979.
- **Unix System III**: atualização da versão 7, que incorporou as características da PWB. Foi um grande sucesso nos anos 80, disponibilizado pela maioria dos fabricantes de microcomputadores. Essa versão foi lançada em 1982.
- **Unix System V**: incorporou melhorias no desempenho e teve ampliada a comunicação entre processos. Foi a primeira versão licenciada pela AT&T com suporte de software. Essa versão foi lançada em 1983.

A AT&T licencia o software Unix, mas não a marca registrada Unix, que é de uso exclusivo da Bell Labs. Dessa forma, o fabricante adquire o código-fonte do sistema, mas não pode utilizar o nome Unix, tendo de atribuir um nome diferente, o que contribui para a proliferação de nomes, gerando confusão em relação às versões. Esses sistemas geralmente são compatíveis entre si, sendo chamados de sistemas Unix-based. Os principais são:

- **AIX**: desenvolvido pela IBM em 1986, inicialmente para a plataforma IBM 6150. Atualmente suporta diversos tipos de arquitetura. Entre suas características podemos destacar sua ferramenta de gerenciamento do sistema, o SMIT.
- **Xenix**: desenvolvido pela Microsoft e posteriormente disponibilizado para a Apple e para o IBM PC, tendo sido uma das versões mais conhecidas do Unix.
- **HP-UX**: desenvolvido em 1984 pela HP, é um dos sistemas Unix mais utilizados atualmente. É baseado na versão System V.
- **SunOS**: desenvolvido pela Sun Microsystems em 1982. Inicialmente, o SunOS era baseado no Unix BSD, mas a partir da versão 5.0 teve seu nome alterado para Solaris e passou a ser baseado no System V Release 4.

Para o desenvolvimento desses sistemas Unix-based, os fabricantes de hardware devem pagar uma taxa de licença para a AT&T, que pode chegar a milhares de dólares por sistema vendido, dependendo do número de usuários que suportam, o que praticamente impossibilitou a utilização do Unix para aplicações pessoais e em empresas de pequeno porte.

Sistemas Unix-based:

- Licenciados pela AT&T: código-fonte do Unix com nome diferente:
  - AIX, Xenix, HP-UX, SunOS etc.
- Versões livres baseadas no Unix BSD:
  - FreeBSD, NetBSD, OpenBSD etc.

Além dessas, surgiram na década de 1990 versões livres de Unix como FreeBSD, NetBSD e OpenBSD, todas baseadas na distribuição BSD. Em reação a essas restrições, começaram a surgir sistemas similares ao Unix que simulavam as características externas do sistema, proporcionando aos usuários as mesmas facilidades e comandos, e permitindo executar os mesmos programas. Esses sistemas não utilizam o código-fonte do Unix e podem ser vendidos sem pagar royalties à AT&T. Alguns exemplos desses sistemas são: UNOS, Unetix e Coherent.

Alguns desses sistemas similares ao Unix mostraram grandes variações nas interfaces gráficas, protocolos de redes, utilitários de gerenciamento, administração de periféricos, entre outras. O desenvolvimento de um código-fonte novo, independente da AT&T, e que constituísse de fato um padrão surgiu, então, como uma necessidade.

Similares Unix

- Desenvolvimento de um código-fonte novo, independente da AT&T.
- UNOS, Unetix e Coherent.
- Grandes variações nas interfaces gráficas, protocolos de redes, utilitários de gerenciamento e administração de periféricos.

Andrew Tanenbaum, pesquisador na área de redes de computadores e Sistemas Operacionais, desenvolveu um protótipo de Sistema Operacional baseado no Unix, chamado de Minix, descrito em Operating System: Design and Implementation (1987). O Minix era utilizado em cursos de ciência da computação em diversas universidades, entre elas, a Universidade de Helsinque, onde um estudante chamado Linus Torvalds decidiu escrever seu próprio Sistema Operacional. Seu objetivo era fazer um "Minix melhor que o Minix". Torvalds desenvolveu então os primeiros kernels do Linux, que já processavam alguns programas de interesse geral, e abriu seu código-fonte na internet, onde cada um poderia desenvolver seus próprios recursos sobre um kernel de conhecimento comum. Essa é a filosofia do software livre, a mesma que proporcionou o desenvolvimento acelerado da internet.

A adesão de estudantes, interessados, curiosos, profissionais e até hackers contribuiu para que a primeira versão considerada estável do Linux estivesse pronta para distribuição no final de 1993. O projeto GNU, da Free Software Foundation, uma organização que apoia projetos de desenvolvimento de software livre, e os trabalhos da Universidade de Berkeley, que produziram o BSD Unix, foram duas importantes fontes de desenvolvimento dos atuais utilitários e aplicativos do Linux.

Para pensar

Uma das vantagens de seu código estar disponível na internet é ter vários grupos dedicados a aprimorar ferramentas e recursos para o sistema, fazendo com que este esteja sempre atualizado, seguro, estável e confiável, constituindo uma plataforma de desenvolvimento que se tornou um padrão de fato.

O Linux está em franco crescimento no mercado, disponível em diversos idiomas, inclusive em português, com interfaces gráficas, processadores de texto, sistemas gerenciadores de bancos de dados, suporte a redes de excelente qualidade e uma infinidade de outras aplicações.
Na verdade, existem atualmente diversas distribuições do Linux, que veremos a seguir.

- Linus Torvalds escreveu seu próprio Sistema Operacional. Seu objetivo era desenvolver um "Minix melhor que o Minix".
- Apoio do GNU e da Free Software Foundation.
- Desenvolvimento do BSD Unix.
- Adesão de estudantes, interessados, curiosos, profissionais, hackers etc.

Distribuições Linux

- Como funciona a GPL?
- A empresa obtém gratuitamente o núcleo do Linux, desenvolve pacotes e comercializa-os.
- Licenças de software tradicionais restringem o acesso ao código-fonte dos programas. A GPL garante a liberdade de compartilhar e alterar o software livre, mantendo seu acesso a todos os usuários.

As versões do Linux são obtidas por meio das distribuições de software. As distribuições de software do Linux são mecanismos de fornecimento de software, desenvolvidos pelo projeto GNU da Free Software Foundation (FSF), que seguem a General Public License (GPL). Enquanto as licenças de software tradicionais servem para restringir o acesso ao código-fonte dos programas, evitando o compartilhamento e a alteração destes, a GPL tem como objetivo garantir a liberdade de compartilhar e alterar o software livre, fornecendo acesso irrestrito ao seu código fonte. Os fabricantes protegem seus softwares por intermédio do copyright. Para criar um contraste, o FSF criou um novo nome, que dá a ideia de um conceito oposto ao do direito autoral, o copyleft. Na verdade, o copyleft da FSF se baseia no mesmo instrumento legal utilizado pelos proprietários de software, o copyright, mas a GPL inclui no copyright termos especiais de licenciamento que garantem a liberdade de uso do software. Esses termos conferem a qualquer usuário o direito de utilizar, modificar e redistribuir o software livre ou qualquer software dele derivado.

Por exemplo: a GPL garante a uma empresa obter gratuitamente o kernel do Linux e trabalhar nele, de forma a melhorar os textos das mensagens do sistema ou traduzi-los para um determinado idioma, e a fazer a documentação deste pacote e vendê-lo como uma distribuição Linux.

A partir do copyleft, qualquer usuário pode usar, alterar e redistribuir o software livre.

Muitas distribuições Linux comerciais surgiram dessa maneira, acoplando ao kernel utilitários, interfaces gráficas, aplicações matemáticas e científicas, sistemas gerenciadores de bancos de dados, entre outros, que são colocados no mercado junto com a documentação e as instruções para a instalação do sistema.

A seguir, são apresentadas algumas características das principais distribuições Linux.

Exercício de fixação 1

Entendendo as licenças Unix e GPL

Quais as diferenças entre as licenças do Unix e da GPL do Linux? Tendo como base o enfoque do usuário, dos desenvolvedores de aplicativos, dos fabricantes de hardware e dos distribuidores de software livre:

- Liste as vantagens da GPL.
- Liste as vantagens dos sistemas Unix-based.

Red Hat Enterprise Linux

- Interface gráfica X-Window e o KDE desktop para gerenciamento de janelas.
- Especializações para versões cliente e servidor, e aplicativos como suítes de escritório.

- Gerenciadores de pacotes RPM e YUM automatizam o processo de instalação e atualização do sistema.
- Versátil, aceita diversas opções de configuração de recursos do sistema.
- Patrocínio do projeto Fedora.

A distribuição Red Hat Enterprise Linux incorpora facilidades, como programas de configuração de recursos do sistema, interfaces gráficas, especializações para versões cliente e servidor e aplicativos como suítes de escritório. Com ela, é fornecido o ambiente gráfico X-Window, contendo o gerenciador de janelas KDE, desenvolvido por um projeto de software livre.

Uma grande vantagem dessa distribuição é a facilidade de instalação, excelente para os iniciantes em Linux. Funcionalidades como pacotes pré-compilados e gerenciadores de pacotes como o RPM e o YUM, que automatizam todo o processo de instalação e a atualização do sistema, facilitam o processo de administração.

Para mais informações, acesse http://www.redhat.com e http://fedoraproject.org

A Red Hat está patrocinando o projeto Fedora juntamente com a comunidade formada por desenvolvedores de software, que dão suporte ao desenvolvimento de software livre. O objetivo desse projeto da comunidade Linux é manter um Sistema Operacional completo, de aplicação geral, com o código-fonte totalmente aberto e gratuito.

CentOS

- Compatível com Red Hat Enterprise Linux.
- Instalação simples em modo gráfico ou texto.
- Suporte a vários idiomas, inclusive o português.
- Ambiente gráfico KDE ou Gnome.
- Rico em ferramentas administrativas e gráficas.
- Gerenciamento de pacotes com RPM e YUM.
- Suporte comercial opcional por meio de revendedores.

A distribuição CentOS deriva da distribuição Red Hat Enterprise Linux (RHEL), de acordo com as regras de redistribuição definidas pela Red Hat Enterprise, que são: remoção de softwares proprietários de terceiros, remoção de imagens, logotipos e textos referenciando a Red Hat, desde que não façam parte de notas de copyright, entre outras. O CentOS é distribuído sob a licença GNU/GPL para aplicações com servidores de pequeno, médio e grande porte e é mantido por uma ativa e crescente comunidade de usuários denominada CentOS Project. Além disso, possui diversas vantagens sobre outras distribuições: rápida correção de bugs e vulnerabilidades, grande rede de repositórios para download, várias opções de suporte como chat, IRC, listas de e-mail, fóruns e um FAQ dinâmico e abrangente. Também é possível obter suporte comercial oferecido por empresas parceiras.

Visite http://www.centos.org para saber mais detalhes sobre a distribuição CentOS.

Debian

- Desenvolvido por um grupo diversificado de programadores.
- Preferido pelos programadores mais experientes.
- Pacotes pré-compilados no formato DEB.
- Ferramentas de gerenciamento de pacotes: APT e Aptitude.

A distribuição Debian foi desenvolvida por um grupo de hackers e é o preferido entre os programadores mais experientes. Possui estrutura especial de pacotes de software pré-compilados no formato DEB e ferramentas de gerenciamento de pacotes muito versáteis, como o APT e o Aptitude.

Acesse o endereço http://www.debian.org para saber mais detalhes.

Ubuntu

- Desenvolvido com foco em segurança.
- Edições para desktop e servidores.
- Garantia de atualizações de até 5 anos para as edições LTS e 18 meses para as demais.
- Instalação simples.
- Ferramentas de gerenciamento de pacotes: APT e Aptitude.
- Suporte a vários idiomas, inclusive o português.

A distribuição Ubuntu é mantida por uma comunidade de usuários e é baseada na distribuição Debian. O Ubuntu contém todas as ferramentas necessárias, desde processadores de texto e leitores de e-mails a servidores web e ferramentas de desenvolvimento, além de utilizar os gerenciadores de pacotes APT e Aptitude.

Para mais detalhes, acesse http://www.ubuntu.com (página em inglês).

O Ubuntu é desenvolvido com foco em segurança, disponibilizando atualizações de segurança gratuitas por pelo menos 18 meses para desktops e servidores. Com a versão Long-Term Support (LTS) adquire-se três anos de suporte para desktops e cinco anos para servidores. Não é cobrado nenhum valor pela versão LTS.

Mandriva Linux

- Fusão da Mandrake com a Conectiva.
- Foco no uso doméstico.
- Ambiente gráfico avançado.
- Compatível com os periféricos nacionais, como monitores, teclados e impressoras.
- Instalação muito simples.
- Manuais e textos de apoio traduzidos para o português.
- Baseada na distribuição Red Hat Enterprise Linux.
- Suporte local e facilidades para a instalação do sistema.

É uma distribuição franco-brasileira, mantida pela empresa Mandriva, formada por meio da fusão da empresa francesa Mandrake e da brasileira Conectiva. Seu nome é a fusão dos nomes das duas empresas. Semelhante ao Windows em vários aspectos, é uma distribuição voltada para desktops, tanto para uso doméstico quanto corporativo. Possui várias opções de idioma para instalação, incluindo o português brasileiro.

Para mais detalhes, acesse o site http://www.mandriva.com. No site, há um link para a comunidade de desenvolvedores.

O instalador do Mandriva é um dos mais amigáveis entre as distribuições Linux. As configurações podem ser realizadas tanto em modo gráfico quanto em modo texto, e dispõem de alguns programas de manutenção de sistema chamados de Drakes, entre eles o MouseDrake, para configurar o mouse, o DiskDrake, para configurar as partições de disco rígido, e Drakconnect, para configurar conexões de rede.

Slackware

- Uma das mais populares distribuições Linux.
- Ótimo desempenho e estabilidade, que o indicam para utilização em servidores de redes.
- Permite trabalhar com pacotes nativos da distribuição Red Hat Enterprise Linux.
- Ambiente gráfico X-Window e gerenciador de janelas fvwm, com estilo similar à interface gráfica do Microsoft Windows.
- Pacotes de segurança avisam, via e-mail, sobre alterações no sistema.
- Instalação complexa.

A distribuição Slackware também é uma das mais populares distribuições Linux, com boa penetração de mercado. Seus pontos fortes são o desempenho e a estabilidade, que a indicam para utilização em servidores. Além disso, possui conversor que permite trabalhar com pacotes nativos da distribuição Red Hat Enterprise Linux. Vem com ambiente gráfico X-Window e o gerenciador de janelas fvwm, que possui design similar ao da interface do Microsoft Windows.

Saiba mais sobre a distribuição Slackware no endereço http://www.slackware.com.

Essa distribuição ainda conta com pacotes de segurança que permitem avisar, via e-mail, que uma função de inicialização do sistema ou um script foi alterado e que isso pode causar transtornos ao sistema. Sua instalação é bem mais complexa.

Exercício de fixação 2
Conhecendo as distribuições Linux

Acesse o site das principais distribuições Linux, verifique o enfoque de cada distribuição (uso em servidores, uso em estações de trabalho, uso doméstico etc.), verifique também se as distribuições possuem suporte comercial, listas de discussão, suporte a múltiplos idiomas e plataformas, e examine a forma de obtê-las (via download, compra de CDs ou DVDs etc.).

- Red Hat.
- SlackWare.
- CentOS.
- Debian.
- Mandriva.
- Ubuntu.

A escolha da distribuição Linux

São muitas as distribuições Linux hoje, algumas com enfoque para uso em servidores visando mais segurança e alta disponibilidade, outras para uso em estações de trabalho visando ambiente gráfico amigável, assim como para uso doméstico, jogos, laptops ou outras finalidades. Além disso, algumas fornecem suporte comercial e garantia de atualizações. Antes de escolher uma distribuição é necessário conhecer um pouco sobre cada uma delas (pelo menos as principais) e ter em mente a finalidade da instalação. Para a nossa finalidade acadêmica, vamos escolher uma distribuição que possua algumas das seguintes características:

- Suporte ao nosso idioma.
- Estabilidade no mercado.
- Ambiente gráfico com assistente de instalação.

- Detecção automática de hardware.
- Ambiente gráfico.
- Gerenciador de pacotes.
- Programas utilitários, como editores de texto, navegadores para a internet, editores de imagens, reprodutores de áudio e vídeo.

Para este curso, escolhemos a versão mais atual do CentOS por ser uma versão gratuita baseada no Red Hat Enterprise Linux, que é uma distribuição bastante sólida no mercado.

Razões para utilizar o Linux

- Software livre.
- Flexibilidade para instalação e reinstalação.
- Sem códigos-chave de licenciamento.
- Custo significativamente menor.
- Tendência de mercado.
- Sistema Operacional da família Unix que mais cresce em número de usuários.

Existem diversas razões para a escolha do Linux como Sistema Operacional. Trata-se de um software livre, com maior flexibilidade para instalação e reinstalação, sem a necessidade de utilização de códigos-chave e licenças. No ambiente Linux, os softwares são normalmente gratuitos, o que praticamente elimina o custo com instalações de software. Outro bom motivo é estar atualizado com as tendências do mercado, já que o Linux vem despontando como o Sistema Operacional da família Unix que mais cresce em número de usuários. Instalar um Sistema Operacional tão complexo quanto o Linux não é uma tarefa simples, mas sem dúvida compensa o trabalho. A seguir serão apresentados os requisitos e os passos necessários para a instalação desse Sistema Operacional.

Hardwares suportados

Na URL http://www.linux-drivers.org podemos consultar a lista de hardwares compatíveis mantida pelas principais distribuições Linux. É necessário investigar a compatibilidade de um hardware antes de comprá-lo. Alguns distribuidores possuem uma classificação adicional, listando hardwares e softwares certificados, isto é, que foram oficialmente testados através de um programa de certificação que garante o funcionamento com o Linux.

Como encontrar hardware compatível?

- Acesse http://www.linux-drivers.org.
- Linux Hardware Compatibility List & Drivers: índice de lista de hardwares compatíveis mantida pelas principais distribuições Linux.

Classificações:

- Compatível: suportado pelo kernel do Linux.
- Certificado: garantia de funcionamento por algum distribuidor.
- Driver em versão Alpha ou Beta: o kernel ainda não suporta, mas pode ser compilado e adicionado como um módulo. Não há nenhuma garantia de funcionamento e não é recomendado em ambiente de produção.
- Não suportado: hardwares testados com funcionamento não confirmado.

Nem todo hardware suporta o Sistema Operacional Linux:

- Projetos especiais de hardware para o Microsoft Windows.
- Bibliotecas de drivers cujo código-fonte não é aberto.
- Incompatibilidade acontece mesmo entre os Sistemas Operacionais da Microsoft.
- Drivers de periféricos mais antigos precisam ser reescritos.
- Vantagens do software livre, com a garantia de acesso aos códigos-fonte.

Lista de verificação de hardware

A maior parte dos dispositivos de hardware é detectada automaticamente durante o processo de instalação. Os drivers desses dispositivos já estão incluídos nas mídias de instalação, embora as distribuições Linux não incluam drivers para todos os dispositivos de hardware. O ideal é montar uma lista de verificação de hardware para o computador que será utilizado para a instalação, como a apresentada na tabela da Tabela 1.1.

Dispositivo	Características	Dados coletados
CPU	Tipo e velocidade.	
Memória RAM	Capacidade.	
Teclado	Marca e modelo.	
Mouse	Protocolo, marca, modelo e número de botões.	
Discos rígidos	Marca, modelo e capacidade.	
Drive de CD/DVD	Marca e tipo.	
Placa de rede	Marca, modelo, tipo e velocidade.	
Placa de vídeo	Marca, modelo e quantidade de memória RAM.	
Monitor	Marca e modelo.	
Placa de som	Marca e modelo.	
Dispositivos USB	Marca e modelo.	

Tabela 1.1
Lista de verificação de hardware.

Requisitos mínimos de hardware

Antes de instalar o Sistema Operacional Linux, devemos conhecer algumas informações do nosso hardware. Neste item conheceremos as configurações mínimas de hardware requeridas para que se possa instalar o Linux. O computador deve atender a alguns requisitos mínimos para poder executar o Sistema Operacional Linux.

- **Unidade de Processamento Central (UCP)**: mesmo sendo possível instalar o Linux em computadores com CPU Intel 80386 e versões anteriores, se for utilizado somente o modo texto. Para suporte total, incluindo o ambiente gráfico, é recomendado o uso de processadores Pentium ou posteriores. O Linux também suporta processadores Intel de 64 bits, AMD de 32 e 64 bits, IBM PowerPC e alguns processadores utilizados em computadores de grande porte. O procedimento de instalação que será visto aqui é válido para a família de processadores Intel e AMD de 32 ou 64 bits. Além do modelo do processador, deve-se atender aos requisitos de velocidade de processamento. Para que se obtenha desempenho satisfatório, deve-se ter um processador com velocidade igual ou maior à de um Pentium III de 700 MHz.

- **Memória**: para instalar e utilizar o Sistema Operacional Linux é necessário somente 4 MB de memória RAM, caso seja utilizado somente o modo texto, e 64 MB de memória RAM, se usado o modo gráfico. Mas o recomendado para atingir performance satisfatória é de pelo menos 512 MB de memória RAM.
- **Drives**: para instalar e utilizar o Sistema Operacional Linux é necessário o mínimo de 2 GB de espaço livre no disco rígido. É possível termos dois ou mais sistemas operacionais instalados no mesmo disco rígido, porém, cada um necessita ter seu próprio esquema de particionamento. Além do disco rígido, é conveniente ter um drive de CD/DVD para facilitar a instalação do sistema.

Neste item veremos como é importante documentar todas as informações do sistema que serão solicitadas durante o processo de instalação do Linux, e também as informações sobre as configurações pré-existentes, no caso de migração do Windows para o Linux.

Instalando o Linux

Neste item veremos como criar uma máquina virtual para instalarmos o Linux e o passo a passo do programa de instalação.

Criação da máquina virtual

O software utilizado para a criação e gerenciamento de máquinas virtuais é o Virtual Box. A seguir são apresentados os passos para fazer a criação da máquina virtual que será utilizada para a instalação do CentOS.

- **Nome da VM e Tipo de Sistema Operacional**: no campo “Nome”, preencher com o valor “CentOS - ADS-001”. No campo “OS Type”, selecionar “Linux” na opção “Sistema Operacional” e “Red Hat (64 bit)” na opção “Versão”.
- **Memória**: selecionar 1 GB de memória RAM.
- **Disco Rígido Virtual**: marcar as opções “Disco de Boot” e “Criar novo disco rígido”.
- **Assistente de criação de discos virtuais**: marcar a opção “VDI (VirtualBox Disk Image)”.
- **Detalhes do armazenamento de disco virtual**: marcar a opção “Dinamicamente alocado”.
- **Localização e tamanho do arquivo de disco virtual**: no campo “Localização”, preencher com o valor “CentOS - ADS-001” e, no campo “Tamanho (S)”, selecionar 15,1 GB de espaço em disco.
- **Sumário**: essa tela apresentará as opções selecionadas durante a criação do disco virtual. Confirme se os dados estão corretos e em seguida clique no botão “Criar”.

Figura 1.5 Iniciando a máquina virtual.

Para iniciar a máquina virtual, basta selecioná-la, como mostra a Figura 1.5, e clicar no botão "Iniciar". Na primeira vez que a máquina virtual é iniciada, o assistente de primeira execução é automaticamente executado. Esse assistente possibilita a seleção da mídia de instalação do Sistema Operacional na máquina virtual. Nessa etapa deve ser selecionado o drive de CD/DVD do hospedeiro. Após a seleção do local onde está inserida a mídia de instalação, é exibida uma tela com os dados selecionados. Confirme se estão corretos e em seguida clique no botão "Iniciar".

Programa de instalação do Linux

Este item mostra passo a passo como instalar a distribuição CentOS. Com base nele, o aluno terá condições de instalar qualquer outra distribuição, pois, de um modo geral, mesmo que algumas etapas apareçam em sequência diferente, as informações necessárias deverão ser as mesmas. Para iniciar a instalação do Linux utilizando um DVD, primeiramente é necessário reconfigurar o BIOS do PC para que o boot seja feito através do drive do DVD. Para isso, devemos entrar na tela de configuração do BIOS e alterar a ordem de inicialização, de modo que o drive de DVD seja o primeiro na ordem de prioridade de boot.

- **Teste de integridade da mídia de instalação**: a primeira tela do programa de instalação apresenta a opção de testar a integridade da mídia de instalação. Geralmente não é necessário executar esse teste. Para passar adiante sem fazer o teste de integridade, selecione a opção "Skip".
- **Iniciando a instalação**: se o computador ou máquina virtual possuírem pelo menos 640 MB de memória RAM, o ambiente gráfico de instalação será aberto por padrão. Caso contrário, uma mensagem informará que a quantidade de memória existente é insuficiente para a instalação no modo gráfico e iniciará a instalação no modo texto.

- **Seleção do idioma**: essa tela mostra os idiomas disponíveis para o processo de instalação. Escolha “Portuguese (Brazilian) (Português (Brasil))”.
- **Seleção do teclado**: em seguida deve ser selecionado o modelo do teclado. Com base no idioma selecionado antes, o instalador sugere o modelo mais apropriado, que neste caso é o “Português Brasileiro (ABNT2)”.
- **Selecionando o tipo de dispositivo que será utilizado na instalação**: esta etapa apresenta duas opções: “Basic Storage Devices”, que inclui discos IDE, SATA, SCSI etc. e “Specialized Storage Devices”, que inclui dispositivos de armazenamento mais complexos como Storage Area Networks (SANs). Selecione a opção “Basic Storage Devices”.
- **Inicializando o disco rígido**: neste momento o instalador analisa o disco rígido e caso não encontre nenhum sistema de arquivos, emitirá um aviso solicitando a inicialização do disco. Esse processo apagará todas as informações existentes no disco. Se a instalação estiver sendo feita em uma máquina virtual, não se preocupe, pois essa ação não apagará nenhuma informação no hospedeiro. Clique no botão “Sim, descarte qualquer dado”.
- **Configuração da rede**: nesta etapa podem ser utilizados dois tipos de configuração:
- **Configuração automática via Dynamic Host Configuration Protocol (DHCP)**: essa é a opção padrão, onde deve ser informado somente o nome da máquina. As outras informações necessárias para integração à rede e acesso à internet, como endereço IP, máscara de rede, gateway, servidores de DNS etc. são fornecidas automaticamente pelo servidor DHCP da rede.
- **Configuração manual**: para configurar a rede manualmente, é preciso clicar no botão “Configure Network”. Nesse caso, é necessário possuir um endereço IP, máscara de rede, endereço IP do gateway e dos servidores de DNS para integração à rede e acesso à internet.
- **Seleção do Fuso Horário**: nesta etapa é definido o fuso horário de acordo com a localização física do sistema. Selecione a opção “América/São Paulo” e deixe marcada a opção “O relógio do sistema utiliza o UTC”.
- **Senha de Administrador (usuário root)**: o administrador do sistema possui plenos privilégios, podendo instalar, alterar ou remover qualquer programa, arquivo ou diretório. A escolha dessa senha deve ser criteriosa, pois caso alguém a descubra, poderá causar sérios danos ao sistema.
- **Tipo de particionamento**: nesta etapa é definido o tipo de particionamento que será utilizado. Selecione a opção “Create Custom Layout”.
- **Particionando o disco rígido**: após inicializar o disco, o programa de instalação abre o gerenciador de partições. Caso o disco tenha sido inicializado anteriormente, ele será exibido sem nenhuma partição, mostrando todo o espaço livre existente. Ao menos duas partições são necessárias para o funcionamento do Linux: uma para o sistema e outra para a área de memória virtual (swap). As principais vantagens de se particionar um disco são: segurança, devido ao isolamento de falhas, e independência entre as partições, evitando problemas no sistema caso a capacidade de alguma partição se esgote. É importante ressaltar que o dimensionamento das partições deve ser feito com bastante critério, para que não ocorram situações onde o tamanho de uma partição se esgote, enquanto outras partições ainda possuam bastante espaço disponível. Nesta instalação serão criadas as seguintes partições:
- **/**: essa partição é a base de todo o sistema, sendo conhecida como partição raiz e simbolizada pela barra (/). Nesta instalação será alocado 5 GB de espaço em disco para a partição raiz, que usará o tipo de sistema de arquivos ext4.

- **/boot**: é nessa partição que fica localizado o kernel do Linux. Essa partição precisa ser primária, pois será acessada pelo BIOS para fazer a carga do Sistema Operacional. Nesta instalação será alocado 100 MB de espaço em disco para a partição /boot. Essa partição utilizará o tipo de sistema de arquivos ext4.
- **/var**: é nessa partição que ficam arquivos de sistema, como arquivos de log, e-mail etc. Nesta instalação será alocado 3 GB de espaço em disco para a partição /var. Essa partição utilizará o tipo de sistema de arquivos ext4.
- **/usr**: é nessa partição que os programas, manuais etc. são instalados. Nessa instalação será alocado 5 GB de espaço em disco para a partição */usr*. Essa partição utilizará o tipo de sistema de arquivos ext4.
- **swap**: essa partição é utilizada como memória virtual e não possui ponto de montagem. Seu tipo de sistema de arquivos é swap e seu tamanho deve ser o dobro da quantidade de memória RAM instalada no sistema. Nesta instalação serão alocados 2 GB de espaço em disco para a partição swap.

A Figura 1.6 mostra a tela onde é selecionado o tipo de partição que será criado. Nesta instalação serão utilizadas somente partições padrão.

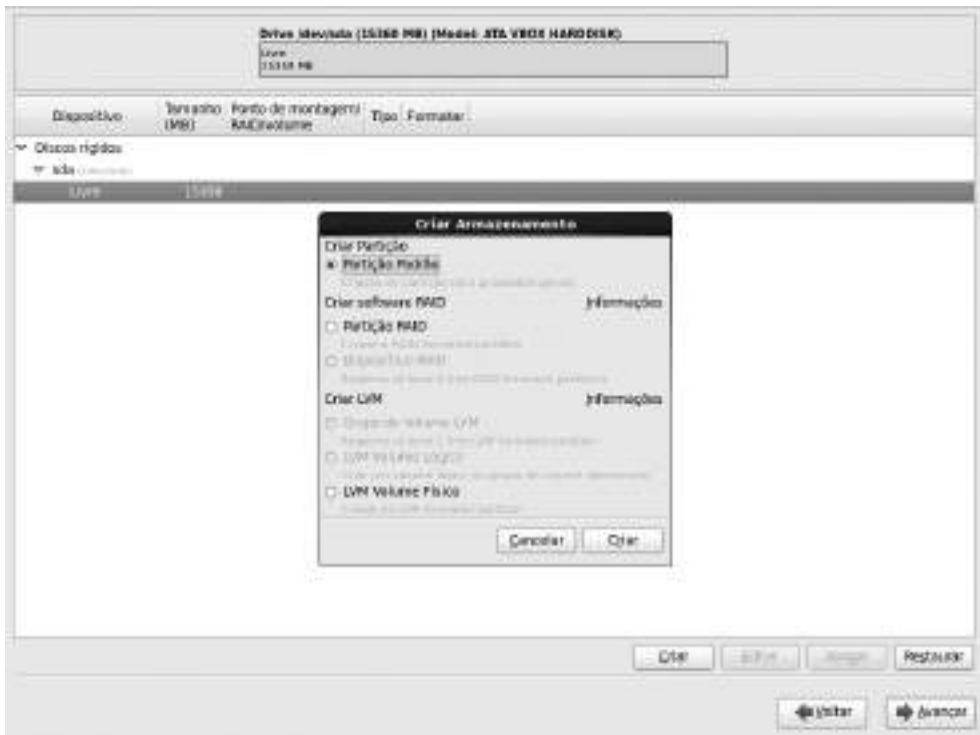


Figura 1.6
Escolha do tipo de partição no disco rígido.

Outras partições podem ser criadas dependendo do tipo de aplicação a que o servidor se destina, assim como pode ser desnecessária a criação de algumas das partições utilizadas no esquema citado anteriormente em determinados casos.

É importante ressaltar que o BIOS suporta apenas quatro partições primárias. Sendo assim, se mais partições forem requeridas, três partições deverão ser criadas como primárias e a quarta deverá ser do tipo estendida, que pode ser subdividida em até 255 partições lógicas. Uma partição estendida não pode conter dados, mas somente partições lógicas. Uma partição lógica, por sua vez, não pode ser utilizada para iniciar a carga de um Sistema Operacional.

A Figura 1.7 mostra a tela de criação de partições, com suas opções. O campo "Ponto de montagem" define o local da árvore de diretórios onde a partição será montada.

O campo “Tipo de sistema de arquivos” define o tipo de sistema de arquivos que será utilizado para formatar a partição. O campo “Tamanho (MB)” define o tamanho da partição em megabytes. Além desses campos básicos, temos a opção de definir a partição como primária e optar pela criptografia dos dados que serão armazenados.

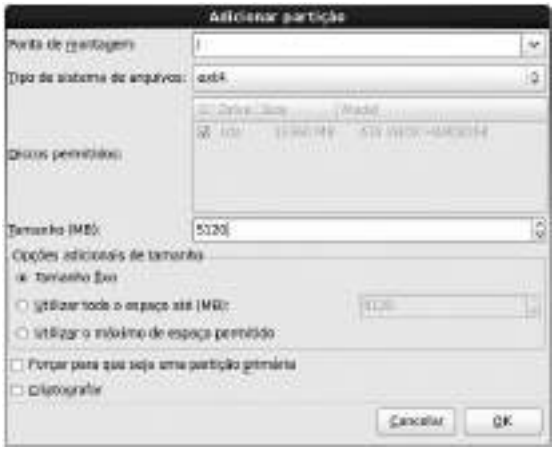


Figura 1.7
Opções para a criação de partições.

- **Gerenciador de boot**: nessa etapa é definido o local do disco onde será instalado o gerenciador de boot (boot loader) ou módulo inicializador do sistema. Os principais gerenciadores de boot do Linux são o Grub e o Lilo. O CentOS utiliza por padrão o Grub, que pode ser instalado no MBR ou no setor de boot da partição ativa. Utilizaremos a configuração sugerida pelo programa de instalação, que instala o Grub no MBR. Para isso, basta deixar a opção “Install boot loader on /dev/sda” e clicar no botão “Avançar”. Nessa tela também é possível adicionarmos outros sistemas operacionais que estejam instalados no disco para serem carregados pelo Grub.
- **Pacotes de aplicações**: nessa etapa é definida uma categoria de instalação. O instalador pré-agrupa conjuntos de aplicações classificando-as nas seguintes categorias: Desktop, Minimal Desktop, Minimal, Basic Server, Database Server, Web Server, Virtual Host e Software Development Workstation. Selecione a opção “Desktop”. Além disso, também são definidos os repositórios que serão utilizados para fazer o download dos pacotes. Nesse caso não será necessário incluir nenhum repositório extra.
- **Instalação dos pacotes**: após a formatação das partições, o instalador inicia o processo de instalação dos pacotes, processo que pode demorar cerca de 15 a 30 minutos, dependendo da performance do computador utilizado e das aplicações que serão instaladas. Fim da instalação: terminada a instalação, o programa de instalação ejetará o DVD e dará ao aluno os parabéns por ter finalizado a instalação. É preciso retirar o DVD e clicar no botão “Reinicializar”.

Roteiro de Atividades 1

Atividade 1.1 – Conhecendo as distribuições Linux

1. Qual das distribuições Linux utiliza o gerenciador de pacotes RPM ou YUM?
2. Qual das distribuições Linux vem com o gerenciador de janelas *fvwm*?
3. Qual das distribuições Linux possui pacotes de software pré-compilados no formato DEB?
4. Quais são as ferramentas de gerenciamento de pacotes utilizadas no Debian e no Ubuntu?
5. Cite dois sistemas baseados em Red Hat.

Atividade 1.2 – Obtendo informações para iniciar a instalação do Linux

Colete as informações necessárias para a instalação do Linux se ele fosse ser instalado na máquina de sala de aula e preencha a Tabela 1.1. O hardware dessa máquina é compatível?

Atividade 1.3 – Preparando o ambiente de instalação

Esta atividade é destinada a preparar a instalação do Linux em uma máquina virtual criada com o software Virtual Box, instalado em um computador com o sistema operacional Windows.

Siga os seguintes passos:

1. Crie uma máquina virtual para posteriormente fazer a instalação do CentOS.

- Nome da VM e Tipo de Sistema Operacional – no campo “Nome”, preencha com o valor “CentOS - ADS-001”. No campo “OS Type”, selecione “Linux” na opção “Sistema Operacional” e “Red Hat (64 bit)” na opção “Versão”.
- Memória – selecione 1 GB de memória RAM.
- Disco virtual – marque as opções “Disco de Boot” e “Criar novo disco rígido”.
- Assistente de criação de discos virtuais – marque a opção “VDI (VirtualBox Disk Image)”.
- Detalhes do armazenamento de disco virtual – marque a opção “Dinamicamente alocado”.
- Localização e tamanho do arquivo de disco virtual – no campo “Localização”, preencha com o valor “CentOS - ADS-001” e no campo “Tamanho (S)”, selecione 15,1 GB de espaço em disco.
- Sumário – esta tela apresentará as opções selecionadas durante a criação do disco virtual. Confirme se os dados estão corretos e em seguida clique no botão “Criar”.

Atividade 1.4 – Instalando o Linux

Faça a instalação do CentOS na máquina virtual criada na Atividade 1.3.

1. Clique com o botão direito na máquina virtual criada e “Configurações”.
2. Vá em “Armazenamento”.
3. Em “Árvore de armazenamento, Controller IDE, Vazio”, ao lado direito em atributos clique no ícone de um CD.
4. Escolha a opção “Selecionar uma arquivo de CD/DVD virtual...”.
5. Escolha a imagem do CentOS no desktop e inicie a máquina virtual.
6. Execute o programa de instalação do CentOS.
7. Teste de integridade da mídia de instalação – “Skip”.
8. Seleção do idioma – “Portuguese (Brazilian)”.
9. Seleção do modelo do teclado – “Português Brasileiro (ABNT2)”.
10. Seleção do tipo de dispositivo de armazenamento – “Basic Storage Devices”.
11. Configuração da rede – configuração automática (padrão); inserir somente o nome da máquina.
12. Seleção do fuso horário – selecionar o fuso “América/São Paulo” e deixar marcada a opção “O relógio do sistema utiliza o UTC”.
13. Definição da senha de root – definir a senha de root.
14. Seleção do tipo de particionamento – “Create Custom Layout”.
15. Criação das partições; crie as seguintes partições:
    - */*, com 5 GB de espaço e tipo *ext4*.
    - */boot*, com 100 MB de espaço e tipo *ext4* (definir como partição primária).
    - */var*, com 3 GB de espaço e tipo *ext4*.
    - */usr*, com 5 GB de espaço e tipo *ext4*.
    - *swap*, com 2 GB de espaço.
16. Instalação do carregador de boot – selecionar a opção “Install boot loader on /dev/sda”.
17. Seleção do tipo de instalação e dos repositórios – selecione a opção “Desktop”. Não é preciso adicionar nenhum repositório.
18. Finalização da instalação – retire a mídia de instalação e clique no botão “Reinicializar”.

Atividade 1.5 – Inicializando o sistema pela primeira vez

Inicialize o sistema pela primeira vez e faça login e logout nele utilizando a conta criada na inicialização. Exercite também outras opções além do logout, como reinicialização e término do sistema.

2

Utilização do sistema

objetivos

Conhecer as funcionalidades da interface gráfica X-Window e dos ambientes de desktop GNOME e KDE.

conceitos

Configurações iniciais, ambiente gráfico, GNOME desktop e KDE desktop.

Configurações iniciais

Após o término bem-sucedido da instalação do Linux e a reinicialização do sistema a partir do disco rígido, o Sistema Operacional será carregado pela primeira vez em sua máquina. A primeira inicialização disponibiliza um serviço de configuração do sistema, em que são mostradas diversas janelas de configuração que serão descritas a seguir.

Informações da licença

Nesta etapa são informadas as regras de licenciamento. É preciso concordar com o acordo de licença para prosseguir.

Criando uma conta de usuário

Assim como em outros Sistemas Operacionais multiusuário, as contas de usuário no Linux servem para identificar e dar permissões aos usuários, limitando seus acessos e privilégios para a utilização do sistema. Deve-se utilizar a conta criada nesta etapa para acessar o sistema. A conta root só deve ser utilizada quando for preciso realizar operações administrativas. Essa medida evita modificações acidentais no sistema, como a remoção inadvertida de arquivos. Crie uma conta especificando: nome do usuário (username), nome completo do usuário e senha.

Configurando data e hora

Permite a definição da data e da hora do sistema ou a definição de um servidor da rede para atualização e sincronização do relógio do sistema através do protocolo Network Time Protocol (NTP).

Saiba mais

O NTP permite manter o relógio de um computador com a hora sempre certa e com grande exatidão. Originalmente idealizado por David L. Mills, da Universidade do Delaware, o NTP foi utilizado pela primeira vez antes de 1985, sendo ainda hoje muito popular e um dos mais antigos protocolos da internet.

Habilitação e configuração do Kdump

O Kdump é um utilitário que grava informações que estavam na memória quando ocorrem falhas no sistema. Ele pode ser útil na detecção de falhas. Nesta etapa é possível habilitá-lo e configurar a quantidade de memória que será alocada a ele.

Ambiente gráfico

Servidor X (X Server):

- X Window System, X-Window, X11 ou simplesmente X, é um protocolo, e seu software associado possibilita o emprego de uma interface gráfica com o conceito de janelas. Originalmente chamado de X, foi desenvolvido no MIT em 1984. Atualmente está na versão 11 e por isso carrega no nome esse número.
- X-Window é o toolkit e o protocolo padrão para interfaces gráficas nos sistemas Unix e assemelhados, como o Linux, embora existam versões para outros Sistemas Operacionais, como o MS Windows e o Mac OS, por exemplo.
- É um software cliente-servidor.
- No CentOS o servidor X utilizado é o X.Org. Outras distribuições utilizam o XFree86.
- Permite acesso via rede, isto é, de um computador é possível capturar a tela de outro computador e trabalhar remotamente.
- Pode ser iniciado automaticamente, ou através da linha de comandos utilizando, por exemplo, o comando *startx*.

O servidor X é o programa que provê a interface gráfica do usuário. Também chamado de X-Window, permite a execução de programas e aplicações gráficas e é compatível com várias plataformas da família Unix. O X-Window pode funcionar em rede, permitindo executar um programa em um computador e ver sua saída gráfica na tela de outro computador conectado à rede.

O X-Window é responsável pelos desenhos de seus objetos na tela e capta as entradas de dados por meio do teclado e do mouse, relacionando-as com as telas gráficas. Para organizar o desktop em objetos como janelas e menus, utiliza um componente chamado "gerenciador de janelas", que integra as aplicações ao ambiente gráfico.

Muitos usuários Linux adotaram o XFree86 ou o X.Org, softwares livres compatíveis com o X-Window, disponíveis nas distribuições Linux.

Iniciando e finalizando o Servidor X

O Servidor X pode ser automaticamente iniciado por intermédio do programa X Display Manager (XDM). O XDM pode ser iniciado automaticamente durante o boot do sistema e mantém o Servidor X rodando, abrindo telas de login e iniciando sessões de trabalho. O Servidor X também pode ser iniciado manualmente pelo usuário através dos comandos *xinit* ou *startx*.

Em caso de mau funcionamento, podemos finalizar o Servidor X pressionando simultaneamente as teclas "Crtl+Alt+Backspace", porém, isso não é recomendável durante a operação normal do sistema, pois o término abrupto de aplicações pode causar perda de dados.
A melhor forma de finalizar o Servidor X é utilizando a opção de saída ou logout existente no gerenciador de janelas, onde é possível escolher entre desligar o computador, reiniciá-lo ou simplesmente fechar a sessão de trabalho.

Configurando o Servidor X

O Servidor X.Org suporta vários mecanismos para suprir e obter configurações e parâmetros dinâmicos do sistema. A ordem de precedência dos mecanismos é:

1. Opções passadas através da linha de comando.
2. Variáveis de ambiente.
3. Opções definidas nos arquivos de configuração *xorg.conf*;.
4. Autodetecção.
5. Valores defaults do sistema.

Arquivo xorg.conf

O X.Org não possui um utilitário de configuração. Em ves disso, ele mesmo tenta gerar uma configuração válida com base em uma série de autodetecções. É possível executar o comando "XOrg – configure" para gerar um arquivo de configuração. O arquivo de configuração do X.Org, o *xorg.conf*, fica localizado no diretório */etc/X11* e é dividido em sessões, que são descritas abaixo.

- **Files**: possui os paths completos para arquivos que o servidor X necessita para funcionar, como tipos de fontes.
- **ServerFlags**: possui opções de configuração globais do servidor X.
- **Module**: é responsável pelo carregamento dinâmico de módulos do servidor X.
- **InputDevice**: possui configurações de dispositivos de entrada de dados, como teclados e mouses. Deve existir um bloco InputDevice para cada dispositivo de entrada.
- **Device**: possui configurações de dispositivos de vídeo. Deve existir pelo menos um dispositivo de vídeo instalado.
- **Monitor**: possui configurações de monitores de vídeo. Deve existir pelo menos um monitor de vídeo instalado.
- **Modes**: é opcional e possui configurações sobre modos de tela disponíveis para o(s) monitor(es).
- **Screen**: possui configurações de dispositivos e monitores de vídeo.
- **ServerLayout**: agrega as sessões Screen e InputDevice.
- **DRI**: é opcional e possui informações sobre a infraestrutura de renderização direta (DRI).
- **Vendor**: possui configurações personalizadas para cada fabricante.

Sessão Files

```
Section "Files"
    FontPath   "/usr/X11/lib/X11/fonts/75dpi/:unscaled"
    FontPath   "/usr/X11/lib/X11/fonts/100dpi/:unscaled"
    FontPath   "/usr/X11/lib/X11/fonts/freefont/"
    FontPath   "/usr/X11/lib/X11/fonts/latin2/"
    FontPath   "/usr/X11/lib/X11/fonts/local/"
```

```
    FontPath   “/usr/X11/lib/X11/fonts/sharefont/”
    FontPath   “/usr/X11/lib/X11/fonts/TTF/”
EndSection
```

Sessão Module

```
Section “Module”
    Load    ”i2c”
    Load    ”bitmap”
    Load    ”dri”
    Load    ”extmod”
    Load    ”freetype”
EndSection
```

Sessão InputDevice

```
Section “InputDevice”
    Identifier    ”Generic Keyboard”
    Driver        ”kbd”
    Option        ”CoreKeyboard”
    Option        ”XkbRules”    ”xorg”
    Option        ”XkbModel”    ”abnt2”
    Option        ”XkbLayout”    ”br”
    Option        ”XkbVariant”    ”abnt2”
EndSection
Section “InputDevice”
    Identifier    ”Configured Mouse”
    Driver        ”mouse”
    Option        ”CorePointer”
    Option        ”Device”        ”/dev/input/mice”
    Option        ”Protocol”        ”auto”
    Option          ”ZAxisMapping” “4 5 6 7”
EndSection
```

Sessão Device

```
Section “Device”
    Identifier    ”Placa de Vídeo Genérica”
    Driver        ”vesa”
    BusID        ”PCI:1:0:0”
EndSection
```

Sessão Monitor

```
Section “Monitor”
    Identifier    ”Monitor Genérico”
    Option        ”DPMS”
    HorizSync    28-64
    VertRefresh    43-60
EndSection
```

Sessão ServerLayout

```
Section “ServerLayout”
    Identifier    ”Default Layout”
    Screen        ”Default Screen”
    InputDevice    ”Generic Keyboard”
    InputDevice    ”Configured Mouse”
EndSection
```

Sessão Screen

```
Section “Screen”
    Identifier    ”Default Screen”
    Device        ”Placa de Vídeo Genérica”
    Monitor        ”Monitor Genérico”
    DefaultDepth    24
    SubSection “Display”
        Depth        4
        Modes        ”1024×768”
    EndSubSection
    SubSection “Display”
        Depth        8
```

```
        Modes        ”1024×768”
    EndSubSection
    SubSection “Display”
        Depth        15
        Modes        ”1024×768”
    EndSubSection
    SubSection “Display”
        Depth        16
        Modes        ”1024×768”
    EndSubSection
    SubSection “Display”
        Depth        24
        Modes        ”1024×768”
    EndSubSection
EndSection
```

Abrindo uma sessão

Para abrir uma sessão de trabalho ou fazer o login no sistema, é preciso informar um nome de usuário e senha válidos. Na tela de login é possível selecionar o gerenciador de janelas e o idioma a serem utilizados, como mostra a figura a seguir.

Figura 2.1
Fazendo o login no sistema.

GNOME desktop

Após realizar o login no sistema, será exibida a janela principal do gerenciador GNU Network Object Model Environment (GNOME), que é o gerenciador padrão do **CentOS**. O GNOME desktop é composto da tela inteira, incluindo os ícones e painéis. A área vazia da tela é chamada apenas de desktop ou área de trabalho.

CentOS
Distribuição Linux de classe Enterprise derivada de códigos fonte gratuitamente distribuídos pela Red Hat Enterprise Linux e mantida pelo CentOS Project.

Figura 2.2
Janela principal do gerenciador GNOME.

- **Barra de menus**: acesso a aplicativos, locais do computador e da rede e utilitários de personalização do sistema.
- **Atalhos rápidos**: acesso rápido a aplicativos. Os seguintes aplicativos vêm por padrão com atalhos rápidos criados: o navegador Firefox, o cliente de e-mail Evolution e o aplicativo de notas rápidas.
- **Barra de status**: exibe o status das conexões de rede, o nível de volume do autofalante, entre outros.
- **Calendário**: exibe o calendário e a hora do sistema. Clicando sobre ele pode-se ajustar a data e a hora do sistema.
- **Status da conta**: possibilita a inserção de informações a respeito do usuário, como telefones de contato, endereço, página pessoal, entre outros. Também possibilita travar a tela, trocar de usuário ou encerrar a sessão de trabalho.
- **Lista de janelas**: exibe as aplicações que estão sendo executadas.
- **Atalhos da área de trabalho**: atalhos para aplicativos ou locais do computador ou da rede. Por padrão, os seguintes atalhos vêm criados: Computador, Pasta pessoal e Lixeira.
- **Desktop**: é a área de trabalho, onde são exibidas as janelas das aplicações que estão em uso.
- **Áreas de trabalho**: possibilita a utilização de mais de um desktop na mesma sessão de trabalho. É possível alternar entre os dois desktops disponíveis, clicando sobre os eles.
- **Lixeira**: acesso à lista de arquivos que foram excluídos pelo usuário por intermédio de programas gráficos de gerenciamento de arquivos.

Menu Aplicativos

Permite o acesso a diversos programas de forma rápida. Esse menu contém diversas categorias:

- **Acessórios**: aplicações como calculadora, captura de tela, dicionário, entre outras.
- **Desenvolvimento**: aplicações para desenvolvimento de software.
- **Escritório**: aplicações para o escritório, como processador de texto, planilha eletrônica, apresentação de slides etc.
- **Gráficos**: programas editores e visualizadores de imagens.
- **Internet**: navegadores, clientes de e-mail, programas de bate-papo, entre outros.
- **Jogos**: jogos dos mais variados estilos.
- **Multimídia**: programas reprodutores de arquivos de áudio e vídeo.
- **Sistema**: utilitários para análise de discos, verificação de atualizações, emulador de terminal, monitor de recursos etc.

Menu Locais

Permite o acesso a locais pré-definidos e a alguns outros recursos do sistema.

- **Pasta pessoal**: mostra os arquivos e pastas do diretório home do usuário, usando o gerenciador de arquivos Nautilus.
- **Mostrar área de trabalho**: exibe o conteúdo da área de trabalho através do Nautilus.
- **Documentos**: mostra a pasta documentos, localizada no diretório home do usuário.
- **Música**: exibe a pasta música, localizada no diretório home do usuário.
- **Imagens**: mostra a pasta imagens, localizada no diretório home do usuário.
- **Vídeos**: exibe a pasta vídeos, localizada no diretório home do usuário.
- **Downloads**: mostra a pasta downloads, localizada no diretório home do usuário.
- **Computador**: exibe os dispositivos de armazenamento do computador.
- **Rede**: mostra os servidores disponíveis na rede.
- **Conectar ao servidor**: conecta a um servidor remoto, utilizando protocolos como FTP, SSH, HTTP, entre outros.
- **Pesquisar por arquivos**: procura por pastas e arquivos no computador.
- **Documentos recentes**: exibe uma lista com os documentos abertos recentemente.

Menu Sistema

Permite o acesso a ferramentas para administração e configuração do sistema e do ambiente computacional.

- **Preferências**: acesso a diversos itens de configuração, como vídeo, teclado, gerenciamento de energia, entre outros.
- **Administração**: acesso a diversas ferramentas de administração do sistema, como configuração de regras de firewall, atualização de software, gerenciamento de serviços, entre outros.
- **Ajuda**: abre o aplicativo de ajuda do GNOME.
- **Sobre este computador**: acesso a diversas informações sobre o computador, como processos em execução, monitoramento de recursos de hardware, entre outros.
- **Bloquear tela**: bloqueia a sessão de trabalho atual. Para desbloqueá-la, o usuário deve informar sua senha.
- **Encerrar sessão**: finaliza a sessão de trabalho atual.
- **Desligar**: desligar ou reiniciar o computador ou, ainda, entrar no modo de hibernação.

Aplicações

Emulador de Terminal: xterm

O xterm é um emulador de terminal de linha de comando. Ele executa funções do sistema utilizando comandos sem sair do ambiente gráfico. A Figura 2.4 mostra a tela do terminal do xterm, que é o emulador de terminal padrão do GNOME.

Figura 2.3
Terminal do sistema GNOME.

Navegador de arquivos

O navegador de arquivos possui um painel lateral e barras de ferramentas, localização e status, que podem ser habilitados ou desabilitados através do menu "Ver". Possui também três modos de exibição: ícones, lista e compacta. Com o navegador de arquivos é possível apagar, mover, copiar ou renomear arquivos selecionados, clicando e arrastando ou escolhendo uma opção no menu. A Figura 2.4 mostra a tela do navegador de arquivos do GNOME.

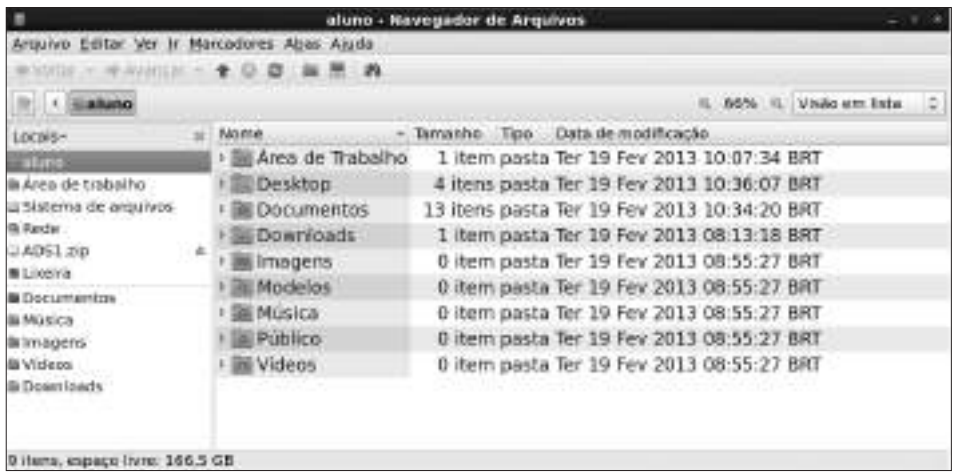


Figura 2.4
Navegador de arquivos do GNOME.

Configurando o GNOME

Para adicionar um ícone no painel do GNOME, basta clicar com o botão direito do mouse sobre o painel e selecionar a opção "Adicionar ao painel". A janela da Figura 2.5 apresenta os itens que podem ser adicionados ao painel.

Figura 2.5
Adicionando um ícone ao painel.

Para configurar a aparência do desktop e das aplicações, basta clicar com o botão direito do mouse sobre o desktop e selecionar a opção “Alterar plano de fundo”. A janela da Figura 2.6 é então exibida, possibilitando a alteração do tema atual, do plano de fundo e das fontes utilizadas no desktop e nas aplicações.

Figura 2.6
Preferências de aparência.

KDE desktop

Como vimos anteriormente, antes de fazer o login, podemos escolher o gerenciador de desktop que vamos utilizar. Os gerenciadores GNOME e KDE são bastante semelhantes, tendo em vista que ambos se propõem a realizar as mesmas funções e são manipulados de forma bastante semelhante. Para utilizar o KDE, o usuário deve selecioná-lo na tela de login. O KDE desktop é a tela inteira, incluindo os ícones e painéis. O espaço na tela onde não há ícones ou painéis é chamado de desktop ou área de trabalho.

O KDE possui facilidades de configuração semelhantes ao GNOME: podemos adicionar e remover ícones do painel do KDE da mesma forma, novos painéis podem ser criados e arrastados para qualquer canto da tela e novos ícones podem ser adicionados da mesma forma como no painel principal.

> Também é possível configurar a aparência do desktop e das aplicações, utilizando a opção “Desktop Settings”.

A Figura 2.7 mostra a tela inicial do KDE desktop e seus elementos, que serão descritos a seguir, bem como suas principais funcionalidades.

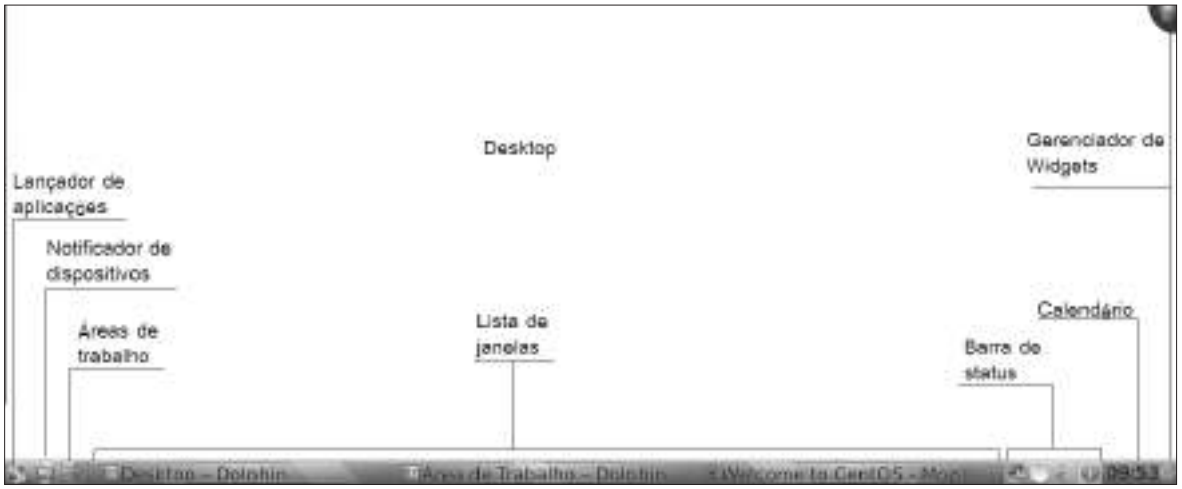


Figura 2.7 KDE Desktop e suas principais informações.

- **Notificador de dispositivos**: gerencia dispositivos removíveis como pen-drives, HDs externos, entre outros.
- **Lista de janelas**: exibe as aplicações que estão sendo executadas.
- **Desktop**: é a área de trabalho, onde são exibidas as janelas das aplicações que estão em uso.
- **Gerenciador de widgets**: adiciona à área de trabalho pequenos aplicativos conhecidos como widgets. Também possibilita a configuração de elementos do desktop.
- **Lançador de aplicações**: acesso às aplicações instaladas, arquivos do usuário e favoritos.
- **Áreas de trabalho**: possibilita a utilização de vários desktops na mesma sessão de trabalho. É possível alternar entre os quatro desktops disponíveis clicando em seus ícones.
- **Barra de status**: exibe o status das conexões de rede, o nível de volume do autofalante, entre outros.
- **Calendário**: exibe o calendário e a hora do sistema. Clicando sobre ele pode-se ajustar a data e a hora do sistema.

Lançador de aplicações

O lançador de aplicações facilita o acesso às aplicações instaladas, arquivos do usuário e favoritos. A Figura 2.8 mostra o lançador de aplicações e suas abas, que serão descritas a seguir.

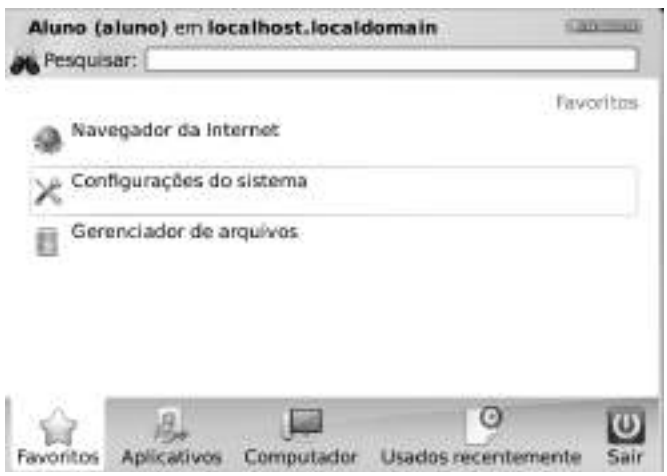


Figura 2.8 Lançador de aplicações.

Favoritos

A aba "Favoritos" contém as aplicações e locais os favoritos do usuário. As aplicações mais utilizadas e os locais mais acessados passam a fazer parte automaticamente dos favoritos. Para incluir uma aplicação aos favoritos, basta clicar sobre seu ícone com o botão direito do mouse e clicar na opção "Adicionar aos favoritos". Por padrão, a lista de favoritos contém o navegador Konqueror, um atalho para as ferramentas de configuração do sistema e o gerenciador de arquivos dolphin.

Aplicativos

A aba "Aplicativos" possui diversas categorias de aplicações que serão descritas a seguir:

- **Administração**: ferramentas de administração do Sistema Operacional. Entre essas ferramentas, podemos destacar: configuração do firewall, gerenciamento de serviços, gerenciamento de contas de usuários, entre outras.
- **Configurações**: diversos utilitários de configuração, como: configuração de impressora, teclado, mouse, rede, entre outros.
- **Desenvolvimento**: aplicações para desenvolvimento de software.
- **Escritório**: um conjunto de aplicações para o escritório, como processador de texto, planilha eletrônica, apresentações de slides, entre outras.
- **Gráficos**: contêm editores e visualizadores de imagens.
- **Internet**: contém navegadores, clientes de e-mail, programas de bate-papo, entre outros.
- **Jogos**: contém possui jogos diversos.
- **Multimídia**: possui reprodutores de arquivos de áudio e vídeo.
- **Sistema**: contém diversos utilitários de configuração, como: ferramentas de configuração de disco, gerenciador de arquivos, visualizadores de arquivos de log, emuladores de terminal, entre outros.
- **Utilitários**: contém utilitários, como alarmes, calculadora, editores de texto, entre outros.
- **Ajuda**: exibe o aplicativo de ajuda do KDE.
- **Arquivos pessoais**: abre o diretório home do usuário no gerenciador de arquivos dolphin.
- **Procurar arquivos/pastas**: buscas por arquivos e diretórios do sistema.

Computador

A aba "Computador" possibilita ao acesso às mídias de armazenamento instaladas no computador, locais e configurações do sistema. A seguir, são descritas as opções presentes na aba "Computador".

- **Configurações do sistema**: ferramentas de configuração do sistema divididas em categorias, como rede e conectividade, administração do computador, entre outras.
- **Informações do sistema**: informações sobre o sistema, como taxa de utilização de disco e memória, status das conexões de rede, modelo e velocidade da CPU, entre outras.
- **Executar comando**: execução de comandos.
- **Pasta do usuário**: exibe o conteúdo do diretório home do usuário que está logado no sistema.
- **Rede**: exibe conexões de rede, diretórios remotos etc.
- **Raiz**: exibe o conteúdo do diretório raiz ("/").
- **Lixo**: exibe o conteúdo da lixeira.

Usados recentemente

A aba “Usados recentemente” exibe as aplicações e os documentos que foram abertos recentemente pelo usuário.

Sair

A aba “Sair” possui opções para finalização da sessão de trabalho e do computador.

- **Sair**: finaliza a sessão de trabalho atual.
- **Bloquear**: bloqueia a sessão. Para desbloqueá-la, o usuário deve informar sua senha.
- **Trocar usuário**: mantém a sessão de trabalho aberta, mas permite que outro usuário faça login no sistema.
- **Suspender para o disco**: pausa o computador sem encerrar a sessão de trabalho atual.
- **Reiniciar**: reinicia o computador.
- **Desligar**: desliga o computador.

Aplicações do KDE

Konqueror

O Konqueror possui funcionalidades de navegador web e gerenciador de arquivos. A Figura 2.9 mostra a funcionalidade de gerenciador de arquivos. Pode-se escolher entre várias formas de exibição de acordo com a preferência do usuário. Os modos de exibição disponíveis são: ícones, detalhes, colunas, emulador de terminal e visualizar o tamanho do arquivo. Pode-se apagar, mover, copiar ou renomear arquivos e diretórios, selecionando-os, clicando e arrastando ou escolhendo uma opção no menu.

Figura 2.9
Konqueror: navegador web e gerenciador de arquivos.

Konsole

Assim como o GNOME, o KDE também possui um simulador de terminal de linha de comando, que é o Konsole. Ele executa funções do sistema utilizando comandos sem sair do ambiente gráfico. A Figura 2.10 mostra a tela do Konsole.

Figura 2.10
Konsole: simulador de terminal de linha de comando.

Configurando o KDE

Para configurar a aparência do desktop, basta clicar com o botão direito do mouse sobre o desktop e selecionar a opção "Configurações da área de trabalho". A janela da Figura 2.11 é então exibida, possibilitando a alteração do tema atual e do papel de parede.

Figura 2.11
Configurando a aparência do desktop.

O aplicativo "Configurações do sistema", exibido na Figura 2.12, possui diversas ferramentas que podem ser utilizadas para configurar o sistema. Essas ferramentas são divididas em categorias e podem configurar desde conexões de rede a aplicações de acessibilidade.

Figura 2.12
Alterando configurações do sistema.

Documentação

A seguir são disponibilizadas referências sobre documentação do GNOME e do KDE.

Documentação on-line:

GNOME: http://library.gnome.org

KDE: http://www.kde.org/documentation

Para documentação local, utilize o comando através de um Xterm:

```
man <nome do comando>
```

Roteiro de Atividades 2

Atividade 2.1 – Conhecendo o ambiente gráfico

1. Identifique o ambiente gráfico que está sendo utilizado.
2. Altere o plano de fundo.
3. Adicione um widget à área de trabalho.
4. Altere o idioma.
5. Adicione o *xterm* ao painel.
6. Altere as configurações de gerenciamento de energia, de modo que o computador hiberne após 2 horas sem uso.
7. Desabilite o serviço de firewall do computador.
8. Ative o outro ambiente gráfico, e execute os passos 2 ao 7; observe a diferença. Qual interface gráfica você achou mais amigável?

Atividade 2.2 – Conhecendo os gerenciadores de arquivos

Utilizando o navegador de arquivos do GNOME ou o Konqueror, crie alguns arquivos utilizando a parte gráfica, navegue pelos diretórios e tente excluir arquivos que você criou, para simular uma situação de erro de operação. Antes disso, não se esqueça de anotar os diretórios em que estes arquivos se encontram, e os nomes dos arquivos. Se não obtiver sucesso na remoção de algum arquivo, tente explicar a razão. Após finalizar a atividade, recupere os arquivos excluídos.

Atividade 2.3 – Trabalhando com arquivos e diretórios

Exercitar algumas ações possíveis com os gerenciadores de arquivos Nautilus e Konqueror, com um conjunto de arquivos existentes, lembrando de retornar à situação anterior:

1. Mova um arquivo para outro diretório.
2. Copie um arquivo para outro diretório.
3. Renomeie um arquivo.
4. Apagar um arquivo.
5. Recuperar o arquivo da pasta *Trash*.

3

Organização do Linux

objetivos

Entender os conceitos de sistemas de arquivos hierárquicos e as características do sistema de arquivos do Linux; conhecer os tipos de arquivos, suas similaridades, diferenças, atributos e facilidades de segurança.

conceitos

Sistema de arquivos do Linux, tipos e atributos dos arquivos, operações com arquivos e diretórios.

Sistema de arquivos do Linux

- Estrutura hierárquica.
- Arquivos sem estrutura.
- Segurança.
- Independência de dispositivo.

O sistema de arquivos do Linux possui diversas características que o tornam seguro e eficients:

- **Estrutura hierárquica**: o sistema de arquivos do Linux possui estrutura hierárquica de diretórios em formato de árvore invertida. Os usuários podem armazenar seus dados sem se preocupar comosestão dispostos fisicamente. O sistema pode ter diversos discos, todos agrupados na mesma árvore de diretórios.
- **Arquivos sem estrutura**: não há estrutura interna, ou seja, não há campos, registros, delimitadores, espaços, cabeçalhos, número de colunas e linha, ou qualquer formato pré-definido a ser seguido pelo usuário ao inserir ou criar o conteúdo do arquivo. O usuário é livre para estruturar e interpretar o conteúdo.
- **Segurança**: os arquivos podem ser protegidos contra o uso não autorizado de diferentes usuários que compartilham o Sistema Operacional.
- **Independência de dispositivo**: o Linux dá tratamento idêntico para arquivos e para dispositivos de entrada e saída. Os mesmos procedimentos e programas utilizados para processar informações armazenadas em arquivos podem ser utilizados para leitura de dados de um terminal, impressão e envio para a entrada de outro programa.

Principais diretórios do Linux e suas funções:

- **/**: diretório-raiz e origem da árvore hierárquica de diretórios.
- **/bin**: binários do sistema utilizado pelos usuários.

- **/boot**: arquivos especiais de inicialização do sistema.
- **/dev**: arquivos especiais de dispositivos de entrada e saída.
- **/etc**: arquivos de configuração, scripts de inicialização de serviços, entre outros.
- **/home**: diretórios pessoais dos usuários do sistema.
- **/lib**: bibliotecas compartilhadas pelos programas e pelo sistema.
- **/mnt**: diretório utilizado como ponto de montagens para dispositivos removíveis, como pen-drives, CDs, DVD, etc.
- **/opt**: diretório utilizado para instalar pacotes opcionais, que não fazem parte da distribuição.
- **/proc**: diretório virtual que contém o sistema de arquivos do kernel.
- **/root**: diretório pessoal do usuário root.
- **/sbin**: comandos de administração do sistema, utilizados pelo usuário root.
- **/tmp**: arquivos temporários do sistema e de programas.
- **/usr**: programas de uso geral do sistema.
- **/var**: arquivos de tamanho variável, como caixas postais de e-mail, cache, lo, etc.

A figura 3.1 representa a estrutura hierárquica de arquivos e diretórios do Linux.

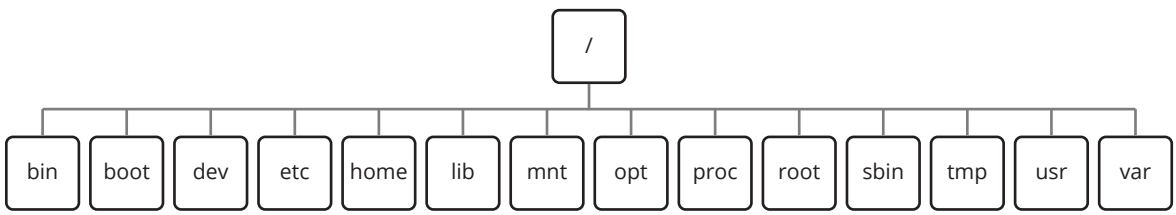


Figura 3.1
Estrutura hierárquica de diretórios do Linux.

Exercício de fixação 1
Conhecendo o sistema de arquivos

Desenhe a árvore do sistema de arquivos até o terceiro nível hierárquico. No terceiro nível, basta listar no máximo três diretórios, se houver. Considerar o ponto de partida, o diretório raiz (/), como o primeiro nível hierárquico.

Exercício de fixação 2
Estrutura de diretórios

O objetivo deste exercício é propor um melhor entendimento da estrutura básica de diretórios do Linux. Relacione os diretórios com seus respectivos conteúdos:

1. - / ( ) Arquivos de configurações locais da máquina.
2. - /bin ( ) Contém diretórios pessoais dos usuários.
3. - /boot ( ) Hierarquia secundária de diretórios.
4. - /dev ( ) Diretório de arquivos temporários.
5. - /etc ( ) Bibliotecas dinâmicas.
6. - /home ( ) Kernel do Linux mais arquivos estáticos do carregador de boot.
7. - /lib ( ) Ponto de montagem para sistemas de arquivos temporários.
8. - /mnt ( ) Comandos executáveis necessários para completar o boot.

9. - /sbin ( ) Diretório de documentação de softwares instalados.

10. - /tmp ( ) Arquivos necessários ao boot geralmente usados pelo root.

11. - /usr ( ) Arquivos de acesso comum.

12. - /usr/Bin ( ) Arquivos de tamanho variável como *spool* e *log*.

13. - /usr/share/doc ( ) Diretório raiz onde começa a estrutura de arquivos inteira.

14. - /var ( ) Arquivos de acesso a dispositivos do sistema.

Montando e desmontando um dispositivo

Para acessar um sistema de arquivos contido em um CD, DVD ou pen drive, é preciso que esses dispositivos estejam montados em alguma partição do disco rígido. Quando o dispositivo é montado, seu estado é informado ao Sistema Operacional. Os ambientes de desktop do **GNOME** e do KDE montam automaticamente esses dispositivos quando são inseridos.

GNOME
GNU Network Object Model Environment é um projeto de software livre abrangendo o ambiente de trabalho GNOME, para os usuários, e a plataforma de desenvolvimento GNOME para os desenvolvedores. O projeto dá ênfase à usabilidade, acessibilidade e internacionalização.

Antes de remover CDs e DVDs, que têm mídias removíveis, devemos desmontá-los para que a gaveta da unidade possa ser aberta. O sistema não permitirá que um dispositivo seja desmontado caso esteja sendo acessado de alguma forma. As mídias removíveis podem ser desmontadas através da linha de comandos ou da interface gráfica.

Quando o sistema é reiniciado através do comando “shutdown”, os dispositivos são automaticamente desmontados.

Inode

O inode é identificado por um número único, que é atribuído a cada arquivo do sistema. Assim, o arquivo de usuários e senhas do sistema pode ser identificado, por exemplo, pelo número 2217, enquanto um arquivo-texto de um usuário pode ser identificado pelo número 456. O Linux organiza os discos rígidos em sequência de blocos. O conteúdo de um arquivo pode ser armazenado em um ou mais blocos, que podem estar espalhados pelo disco e não estarem dispostos de forma sequencial. O inode é uma estrutura de dados, com informações que permitem encontrar os dez primeiros blocos desse arquivo. Além da lista de locais dos dez primeiros blocos, o inode guarda outras informações importantes a respeito de um arquivo, como: tipo, permissões associadas, proprietário, grupo, tamanho, última vez em que foi modificado etc. Quando o arquivo é aberto, o inode fica disponível na memória principal, permitindo, com isso, que os primeiros dez blocos de um arquivo possam ser encontrados rapidamente, pois o acesso à memória é bem mais rápido que ao disco. Assim, o inode serve também para aumentar a eficiência de acesso ao conteúdo de arquivos pequenos no Linux. Quando um arquivo é maior do que dez blocos, o Linux passa a utilizar técnicas de acesso indireto, onde as posições do décimo-primeiro bloco em diante são armazenadas não mais no inode, mas em outro bloco no disco, chamado de “bloco indireto”. As posições dos blocos indiretos ficam armazenadas no inode e, para acessar um bloco indiretamente, passa a ser necessário tanto um acesso ao inode quanto um acesso ao bloco indireto no disco, o que obviamente aumenta o tempo de acesso ao conteúdo e reduz a eficiência de acesso a arquivos grandes (maiores que 10 blocos). Mesmo o acesso indireto é limitado para arquivos muito extensos. Assim, pode ser utilizado o método indireto duplo, onde um bloco chamado de “bloco indireto duplo” contém as posições de outros blocos indiretos e estes, por sua vez, contêm as localizações dos blocos do arquivo. Os blocos indiretos duplos também são armazenados no disco e têm suas posições guardadas no inode.

Para permitir tamanhos ilimitados de arquivos, esse processo pode ser expandido através de uma hierarquia de blocos indiretos, com blocos indiretos únicos, duplos, triplos e, dessa forma, aumentar sem limitações o conteúdo dos arquivos.

Tipos de arquivos

No Linux todo objeto que é manipulado pelo Sistema Operacional é representado por um arquivo, incluindo diretórios, dispositivos de hardware e conexões de rede. Para identificar o tipo do arquivo, o Sistema Operacional consulta as informações contidas em seu inode. A seguir são apresentados os tipos de arquivos existentes no Linux.

Saiba mais

Pode-se utilizar uma extensão para o nome do arquivo, colocando um segundo nível de nome, separado por ponto. Essa medida facilita ao usuário encontrar arquivos, utilizando as extensões dos nomes juntamente com referências ambíguas ou caracteres curinga.

Arquivo regular

A estrutura básica que o Linux utiliza para armazenar informações é o arquivo. Nos arquivos são armazenados todos os tipos de dados, desde textos até instruções em código de máquina. Essa é a estrutura utilizada para armazenar todos os tipos de informações necessárias para a operação do sistema. A identificação dos arquivos por números é utilizada apenas internamente pelo sistema, já que, na prática, além de tediosa, a identificação numérica perde o sentido. Dessa forma, o sistema permite a identificação dos arquivos através de nomes. O nome do arquivo pode ter qualquer sequência de até 256 caracteres, mais do que suficiente para descrever o conteúdo do arquivo, dando um sentido à identificação. Para um sistema com milhares de arquivos, é pouco provável que não sejam escolhidos nomes que já estejam sendo utilizados por outros arquivos.

Caracteres curinga

Têm a função de substituir outros caracteres, pois ao aparecerem numa determinada posição, indicam que os nomes de arquivos que atendem à condição solicitada podem ter quaisquer caracteres naquela posição. O caractere curinga "*" (sem aspas) pode substituir um nome ou um conjunto de caracteres numa determinada posição de um nome de arquivo. Por exemplo: "P*" significa que todos os nomes de arquivos que começam com "P", seguidos de quaisquer conjuntos de caracteres até o tamanho máximo de 256 caracteres, atendem à condição solicitada. O caractere curinga "?" (sem aspas) pode substituir uma letra, em determinada posição do nome, por qualquer caractere. Por exemplo: "P??A" significa que todos os nomes de arquivos com quatro caracteres que começam por "P" e terminam com "A" atendem à condição solicitada.

Diretório

- Conjunto de arquivos.
- Listagem de arquivos com seus inodes correspondentes.

Para o Sistema Operacional, um diretório é apenas um arquivo especial que contém uma listagem de nomes de arquivos e seus inodes correspondentes. O diretório desempenha exatamente a mesma função de um catálogo de telefones: dado o nome de um arquivo, o Sistema Operacional consulta seu diretório e obtém o número do inode correspondente ao arquivo. Com esse número, o sistema de arquivos pode examinar outras tabelas internas para determinar onde está armazenado o arquivo e torná-lo acessível ao usuário. A localização de arquivos é mostrada na Figura 3.2.

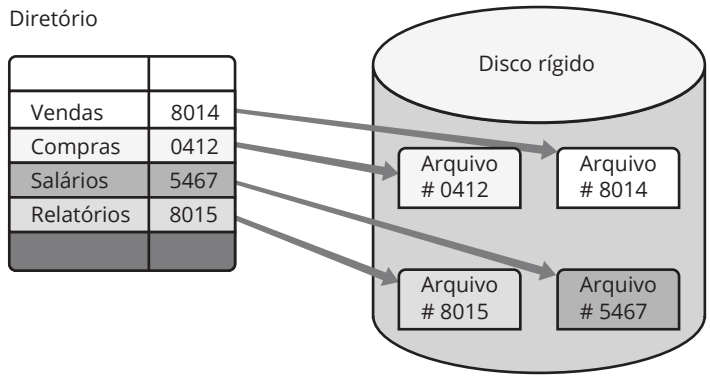


Figura 3.2 Localização de arquivos no disco rígido.

O Linux permite organizar arquivos agrupando-os em diretórios. Um diretório desempenha a mesma função de uma gaveta de armário para arquivamento, agrupando todos os arquivos num lugar comum onde possam ser facilmente encontrados. Assim, o usuário ganha flexibilidade para agrupar arquivos de forma lógica. Por exemplo, ao criar arquivos com os resultados das vendas das filiais de uma empresa, ele pode agrupar esses arquivos em diretórios com o mesmo nome do local da filial. Os diretórios também podem ter nomes compostos por até 256 caracteres. É recomendado também que o nome do diretório faça referência ao seu conteúdo. Cada usuário do sistema tem seu diretório pessoal, que geralmente possui o mesmo nome do usuário.

Arquivos de dispositivos

- Os dispositivos no Linux são referenciados por arquivos.
- Dispositivos orientados a caractere e a bloco de caracteres.
- Os arquivos de dispositivos ficam agrupados no diretório */dev*.

O sistema de arquivos estende o conceito de arquivo para tratar os dispositivos de entrada e saída, como impressoras, unidades de fitas e ainda outros tipos que podem ser instalados em um Sistema Operacional Linux. Os dispositivos são tratados como arquivos especiais e manipulados como arquivos comuns do sistema. Os arquivos de dispositivos podem ser de dois tipos:

- **Arquivos de dispositivos orientados a caractere**: realizam suas transferências de dados byte a byte e de modo sequencial. As portas seriais são exemplos de dispositivos orientados a caractere. Esse tipo de arquivo é representado pela letra "c".
- **Arquivos de dispositivos orientados a blocos de caracteres**: realizam suas transferências de dados em blocos de tamanho que pode variar entre 512 bytes e 32 Kbytes, sendo o acesso feito de modo aleatório. Os discos rígidos e as unidades de fita são exemplos de dispositivos orientados a bloco. Esse tipo de arquivo é representado pela letra "b".

Alguns dispositivos só podem ser acessados no modo caractere, como terminais e impressoras, pois não têm recursos para o acesso bloco a bloco. Outros dispositivos permitem o acesso bloco a bloco, como discos e fitas, mas podem, também, ser acessados caractere a caractere, dependendo da operação efetuada. Por exemplo: na formatação de um disco, os blocos ainda não existem, logo, o acesso inicial a esse dispositivo deve ser orientado a caractere. Existem outras situações em que os dispositivos orientados a blocos podem ser acessados caractere a caractere como, por exemplo, para a execução de cópias de segurança; entretanto, o contrário não ocorre, ou seja, os dispositivos orientados a caractere não podem ser acessados pelo modo bloco a bloco. Por convenção, todos os dispositivos de E/S no Linux recebem nomes individuais de arquivo e são agrupados no diretório */dev*, que é a abreviatura de *devices*. Os dispositivos mais comuns são designados de forma padronizada:

- **/dev/lp**: impressora do sistema.
- **/dev/tty**: terminais ou linhas de comunicação.

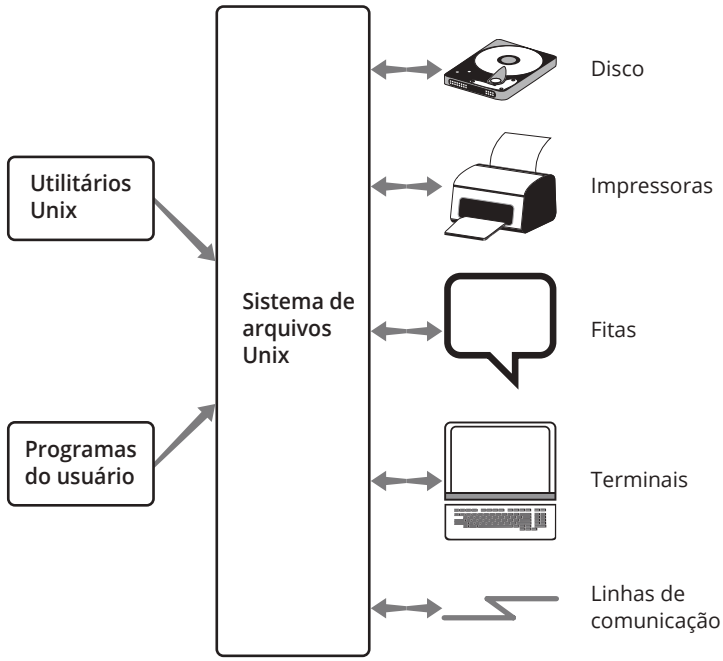


Figura 3.3
Arquivos de dispositivos.

As operações de entrada e saída nesses dispositivos funcionam exatamente da mesma forma que nos outros tipos de arquivos. Os programas de aplicação projetados para funcionar com arquivos podem assim funcionar, com todos os tipos de dispositivos de entra e saída (E/S), sem a necessidade de modificações, característica conhecida como independência de arquivos dispositivos (device files). A independência de dispositivos permite que os mesmos comandos utilizados para manipular arquivos normais possam ser utilizados para executar funções semelhantes com arquivos de dispositivos, conforme o exemplo:

Cópia de um arquivo para outro:

```
# cp vendas relatórios
```

O mesmo comando pode ser utilizado para copiar o arquivo para a impressora:

```
# cp vendas /dev/lp
```

Os dispositivos de entrada e saída (E/S) são tratados como arquivos especiais e manipulados como arquivos comuns do sistema. Podem ser de dois tipos:

- Dispositivos orientados a caracteres: tipo de arquivo representado pela letra “c”.
- Dispositivos orientados a blocos de caracteres: tipo de arquivo representado pela letra “b”.

Named pipes

Permitem que dois processos possam trocar informações por intermédio de um canal bidirecional.

Pipes convencionais são os dados que entram em um extremo do canal saem no outro extremo, definindo um sentido de comunicação.

O Linux provê uma funcionalidade de comunicação entre processos denominada named pipe. O named pipe permite a comunicação bidirecional entre dois processos executados no mesmo Sistema Operacional. Um named pipe é referenciado pelos processos que conecta através de seu nome e faz parte do sistema de arquivos. Quando criamos os named pipes, eles são associados a diretórios e a permissionamentos de acesso, assim como os arquivos regulares.

Os named pipes podem ser criados através dos comandos *mkfifo* ou *mknod* e removidos com o comando *rm* ou por meio da chamada de sistema *unlink*.

Além do named pipe existe também o pipe convencional, representado pelo caractere "|", que é utilizado em comunicações unidirecionais, conectando a saída de um processo à entrada de outro. Os pipes funcionam da mesma forma que os descritores de arquivos padrão, apenas lendo a informação de entrada e escrevendo-a na saída, sem a preocupação com o modo como ela será tratada pelos processos envolvidos na comunicação. A diferença entre o named pipe e o pipe convencional é que, nesse último, os processos conectados devem possuir uma relação de pai para filho ou serem "irmãos". Além disso, o named pipe precisa ser explicitamente encerrado após seu uso, ao contrário do pipe convencional, que é encerrado automaticamente após a execução dos processos que ele conecta.

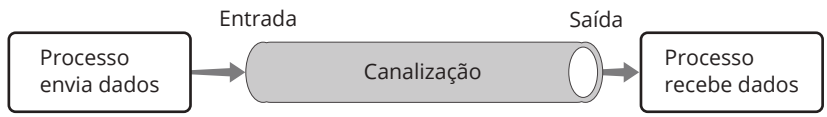


Figura 3.4 Comunicação entre processos utilizando pipes.

Para que servem os links

São ponteiros para outros arquivos, links simbólicos ou *soft links* que funcionam como ponteiros para determinados arquivo, e têm as características de um arquivo.

Vamos supor que o arquivo */usr/local/admin/vendas*scontenha informações de vendas de uma empresa e toda a equipe de vendedores precise acessar este arquivo. Imagine o trabalho que daria copiar este arquivo para o diretório home de cada funcionário e mantê-los atualizados. Com os links simbólicos criamos um link em cada diretório home, que aponta para o arquivo original localizado no diretórios*/usr/local/admin/vendas*, reduzindo o trabalho e mantendo o acesso às informações sempre atualizadas. Cada usuário pode criar seus links com nomes diferentes em seu diretório home, apontando para o mesmo arquivo original. Outra vantagem é simplificar o acesso a arquivos que tenham um caminho (path) extenso, com vários subdiretórios. Ao criar um arquivo do tipo link em seu diretório home, o usuário evita a digitação de todo o caminho do arquivo vendas, por exemplo, manipulando-o diretamente através de seu diretório home. O comando para a criação de um arquivo do tipo link possui a sintaxe:

```
ln -[opções] origem [destino]
```

Onde origem ou destino podem ter um nome de arquivo ou o caminho completo do arquivo na estrutura hierárquica.

Exercício de fixação 3
Links

Pesquise a diferença entre *HardLink* e *SoftLink*.

Sockets

Mecanismos para troca de dados entre processos.

Os processos podem estar sendo executados no mesmo computador ou em computadores diferentes conectados através da rede. Uma vez estabelecida a conexão, os dados podem trafegar nos dois sentidos até uma das pontas encerrar a conexão.

Os sockets são utilizados para a comunicação bidirecional entre dois processos, que podem ser executados no mesmo computador ou em computadores diferentes. Os principais tipos de sockets utilizados no Linux são: Unix domain socket ou Inter Process Communication socket (IPC socket), que é utilizado para a comunicação entre processos executados em um mesmo sistema operacional, e o socket de rede, que é utilizado para a comunicação entre processos executados em computadores diferentes, interligados por uma rede. Existem casos em que os sockets de rede são criados para comunicação entre processos que são executados no mesmo computador.

Entre os sockets de rede, podemos destacar o stream socket e o datagram socket.

Apesar de os arquivos sockets terem diversas funções específicas, são muito similares aos arquivos comuns, sendo tratados da mesma forma pelo sistema de arquivos do Linux. Um socket de rede tem um funcionamento parecido com o telefone, isto é, cada arquivo socket representa um ponto de conexão de uma linha de comunicação, sendo que, entre esses pontos, existe a rede de comunicação de dados. Além disso, também têm seu próprio número de identificação na rede, assim como o telefone na rede telefônica. Para permitir que se estabeleça o contato entre dois sockets, o socket local e o remoto são identificados por um par endereço IP e porta.

Um socket pode ser criado através da chamada de sistema socket e removido através do comando *rm* ou da chamada de sistema *close*, quando ele não estiver mais sendo utilizado. A maioria das aplicações no Linux usa sockets.

Atributos dos arquivos

- **Nome**: nome do arquivo.
- **Localização**: local de armazenamento do arquivo no disco.
- **Tamanho**: tamanho do arquivo em bytes.
- **Ligações**: nomes pelos quais o arquivo é conhecido.
- **Propriedade**: usuário que é o dono (owner) do arquivo.
- **Grupo**: grupo de usuários que acessa ao arquivo.
- **Tipo**: tipo do arquivo.
- **Criação**: data e hora de criação do arquivo.
- **Modificação**: data e hora da última modificação do arquivo.
- **Acesso**: data e hora do último acesso ao arquivo.
- **Permissões**: permissões de acesso ao arquivo.

Os arquivos possuem diversos atributos, que são armazenados em seus inodes correspondentes. Entre esses atributos, podemos destacar:

- **Nome**: nome do arquivo.
- **Localização**: local onde o arquivo está armazenado no disco.
- **Tamanho**: tamanho do arquivo em bytes.
- **Ligações**: nomes pelos quais o arquivo é conhecido.
- **Propriedade**: usuário dono (owner) do arquivo.
- **Grupo**: grupo de usuários que pode ter acesso ao arquivo.
- **Tipo**: tipo do arquivo.
- **Criação**: data de criação do arquivo.
- **Modificação**: data de modificação do arquivo.
- **Acesso**: data do último acesso ao arquivo.
- **Permissão**: permissões de acesso ao arquivo.

Todas essas informações são automaticamente mantidas pelo sistema na medida em que os arquivos são criados e utilizados. Os diversos utilitários usam essas informações para processar arquivos seletivamente. Os utilitários de backup do Linux, por exemplo, podem preservar cópias apenas daqueles arquivos que foram modificados após alguma data específica. A data da última modificação é utilizada para selecionar os arquivos apropriados. Ao listarmos arquivos e diretórios utilizando o comando *ls* com a opção *–l*, visualizamos as informações mostradas na Figura 3.5.

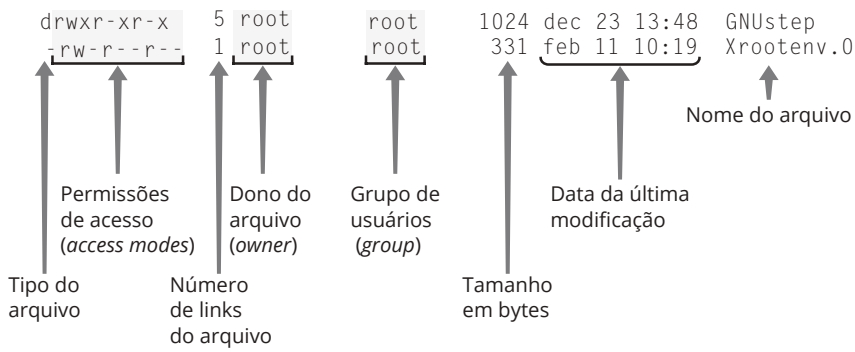


Figura 3.5
Saída do comando *ls* com a opção *–l*.

Permissões de arquivos

O Linux é um sistema projetado para ser multiusuário. Para suportar operações em ambientes com múltiplos usuários, o Linux dispõe de mecanismos que restringem o acesso a arquivos e diretórios, baseados na identificação do usuário que solicita o acesso, e ao modo de acesso atribuído a cada arquivo e diretório.

Todo arquivo e diretório é associado a um usuário que é chamado de dono (owner). O usuário que inicialmente cria o arquivo é o dono do arquivo. Cada usuário pode pertencer a um ou mais conjuntos de usuários que são chamados de grupo. Cada arquivo ou diretório é associado a um grupo, que é atribuído ao arquivo quando este é criado. Da mesma forma, o usuário que inicialmente cria o arquivo ou diretório determina o grupo que pode acessá-lo. Esse grupo associado ao novo arquivo ou diretório é o grupo primário do usuário que os criou. Tanto o dono como o grupo de um arquivo podem ser alterados. As permissões de acesso, conhecidas como modos de acesso, determinam as operações que um usuário pode realizar em um arquivo. A seguir estão os três tipos básicos de permissão que podem ser aplicadas a um arquivo ou diretório.

- **r (read)**: permite acesso apenas para leitura.
- **w (write)**: permite acesso para leitura e gravação.
- **x (execute)**: permite executar o arquivo.

Note que as permissões habilitam a execução de ações diferentes em arquivos e diretórios. Arquivos ou diretórios podem ter uma ou mais permissões: um arquivo que tenha as permissões *rw* pode ter seu conteúdo lido e alterado por um usuário com acesso a ele, mas não pode ser executado por esse usuário, pois o arquivo não tem a permissão de execução x. Os modos de acesso a um arquivo ou diretório consistem em três conjuntos de permissões, com três caracteres cada um. O primeiro conjunto de permissões se refere ao usuário que é o dono do arquivo ou diretório. O segundo conjunto se refere ao grupo de usuários que podem ter acesso ao arquivo. O terceiro conjunto de permissões restringe o acesso a outros usuários (others), exceto o dono ou o grupo associado ao arquivo. Para alterar o dono ou o grupo do arquivo, são utilizados os comandos *chown* (change owner) e *chgrp* (change group), respectivamente:

```
# chown novodono arquivo

# chgrp novogrupo arquivo
```

O comando *chown* também permite alterar dono e grupo de uma só vez:

```
#chown novodono:novogrupo arquivo
```

As permissões de um arquivo também podem ser alteradas através do comando *chmod* (change mode). Cada uma das nove permissões (ler, escrever e executar; para o dono, para o grupo e para os outros) pode ser individualmente concedida ou negada com esse comando. A seguir, são apresentados alguns exemplos de uso do comando *chmod*.

```
# chmod +r arquivo (concede acesso de leitura ao arquivo, ao dono,
ao grupo e aos outros.)

# chmod go-w arquivo (nega acesso de escrita aos membros do grupo
e aos outros.)
```

É possível utilizar uma sintaxe alternativa para o comando *chmod*, com uma codificação que utiliza três números na base octal (de 0 a 7). Cada número octal corresponde a um conjunto *rwx*, que de acordo com a sua posição pode representar as permissões do dono (primeiro conjunto), do grupo (segundo conjunto) e dos outros (terceiro conjunto). Cada um desses números octais corresponde a três bits, sendo o primeiro deles associado à permissão de leitura, o segundo à permissão de escrita e o terceiro à permissão de execução. Se o bit tiver o valor 0, indica ausência de permissão e, se tiver o valor 1, indica a presença da permissão. A codificação completa na base octal tem os seguintes significados:

Tabela 3.1
Sintaxes do comando ***chmod***.

Modo octal	Modo binário	Tipo de permissão
0	000	Sem permissão.
1	001	Permissão de execução.
2	010	Permissão de escrita.
3	011	Permissão de escrita e execução.
4	100	Permissão de leitura.
5	101	Permissão de leitura e execução.

Modo octal	Modo binário	Tipo de permissão
6	110	Permissão de leitura e escrita.
7	111	Permissão total (leitura, escrita e execução).

O exemplo a seguir mostra o comando *chmod* com a sintaxe que utiliza a base octal:

```
# chmod 764 arquivo
```

7: permite acesso à leitura, gravação e execução do arquivo pelo seu dono.

6: permite acesso à leitura e gravação do arquivo pelo grupo.

4: permite somente leitura para os outros usuários.

Correspondentes binários:

- Cada dígito binário corresponde a uma letra: “r”, “w” e “x”:
  0: indica ausência da permissão.
  1: indica concessão da permissão.
- São três números octais, correspondentes a três dígitos binários (0=000 a 7=111). Um número para o dono, um para o grupo e um para outros. Exemplo:
  `# chmod 764 arquivo`
  - Onde: 7 permite leitura, escrita e execução (rwx) para o dono (u), 6 permite leitura e escrita (rw) para o grupo (g) e 4 permite leitura (r) para os outros (o).

A Figura 3.6 mostra as sintaxes padrão e octal, com seus respectivos correspondentes binários.

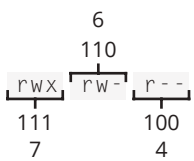


Figura 3.6 Sintaxes do comando *chmod*.

Caso haja uma tentativa de um usuário acessar um arquivo para realizar determinada ação, mesmo ele pertencendo ao grupo ou sendo dono do arquivo, se ele não tiver permissão para realizar a ação pretendida, receberá a mensagem “access denied” (acesso negado).
A Tabela 3.2 mostra as permissões ou modos de acesso para arquivos e diretórios.

Modo	Operação permitida em diretório	Operação permitida em arquivo
R	Listar conteúdo do diretório.	Ler o conteúdo do arquivo.
W	Criar ou remover diretórios e arquivos dentro do diretório.	Alterar o conteúdo do arquivo.
X	Acessar arquivos e subdiretórios dentro do diretório.	Executar o arquivo.

Tabela 3.2 Permissões de acesso para arquivos e diretórios.

Permissões:

- **Dono (owner)**: proprietário do arquivo.
- **Grupo (group)**: grupo primário do dono do arquivo.
- **Outros (others)**: outros usuários que não são donos e que não fazem parte do grupo.

Comandos para manusear permissões:

`# chown novodono arquivo` (muda o dono do arquivo)

`# chgrp novogrupo arquivo` (muda o grupo do arquivo)

`# chmod ugo±rwx arquivo` (muda as permissões do arquivo)

Exercício de fixação 4
Atributos e permissões de arquivos

Que permissão está sendo concedida ao dono, usuário e grupo em relação aos arquivos abaixo?

- Chmod 750 atividade1
- Chmod 000 atividade2
- Chmod ug-x atividade3
- Chmod o+x atividade4
- Chmod 680 atividade5

Especifique os tipos de arquivos seguintes, suas permissões e sintaxes no padrão octal:

- brwxr-x-wx root aluno arquivo1
- c---rw---x aluno aluno arquivo1
- -------rwx root aluno arquivo1
- lrwxrwxrwx root root arquivo1
- drw-rw---- root root arquivo1
- c--------- root root arquivo1
- srwx----wx root root arquivo1
- pr-xr-xr-x root aluno arquivo1

Operações com arquivos e diretórios

Existem diversos comandos que são utilizados para trabalhar com arquivos e diretórios. Esses comandos realizam funções de cópia, busca, remoção, entre outras. A seguir serão apresentados os principais comandos de manipulação de arquivos e diretórios.

Para a criação de arquivos, existem diversas formas de se criar um arquivo no Linux. Veremos algumas dessas formas.

Redirecionamento de entrada e saída

O Linux possui um dispositivo padrão de entrada e outro de saída, que são respectivamente o teclado e o monitor.

- Os sinais "<" e ">" redirecionam a entrada e a saída padrão para um novo arquivo.
- O comando *ls* pode ter sua listagem redirecionada da saída padrão (o monitor) para o novo arquivo "listadearquivos", criando o arquivo se ele não existir.

```
# ls > listadearquivos
```

Toda vez que executamos um comando, digitamos as opções e parâmetros por intermédio do teclado e então recebemos sua resposta no monitor. O Linux possui um dispositivo padrão de entrada e outro de saída, que são respectivamente o teclado e o monitor.

Podemos redirecionar, ou seja, alterar a entrada ou a saída padrão dos comandos utilizando os sinais "<" e ">", respectivamente. Essa característica do Linux é conhecida como redirecionamento de entrada e saída. Dessa forma, os dados provenientes da entrada padrão (o teclado) podem ser direcionados para outro tipo de saída, como um arquivo existente ou um novo arquivo que será criado. Podemos, por exemplo, criar um arquivo que contenha a listagem de todos os arquivos do diretório home do usuário. Para isso, devemos executar o seguinte comando no diretório home desse usuário:

```
# ls > listadearquivos
```

O comando *ls* lista todos os arquivos do diretório, redirecionando essa listagem da saída padrão (o monitor) para o novo arquivo "listadearquivos", criando o arquivo no diretório home do usuário. Na verdade, podemos criar arquivos utilizando diversos comandos, simplesmente redirecionando a saída padrão para um novo arquivo. Podemos utilizar o comando *cat* para concatenar dois arquivos existentes e gerar um novo arquivo, como no exemplo:

```
# cat arquivo1 arquivo2 > arquivonovo
```

O comando criará o arquivo "arquivonovo", com o conteúdo do "arquivo1" seguido do conteúdo do "arquivo2". A maneira mais simples de criar um arquivo de texto é redirecionar a entrada do teclado para o arquivo, através do comando *cat*:

```
# cat > arquivo.txt
```

Após o comando, todo o texto digitado será armazenado no arquivo, pois como a entrada não foi especificada, esta continua sendo a padrão, ou seja, o teclado. Para encerrar a execução do comando, basta finalizar a digitação e teclar "Ctrl+D" duas vezes ou teclar "Enter" e "Ctrl+D".

Podemos utilizar o comando *cat* para a concatenação de arquivos existentes, gerando um novo arquivo:

```
# cat arquivo1 arquivo2 > arquivonovo
```

A maneira mais simples de criar um arquivo de texto é redirecionar a entrada do teclado para o arquivo, por meio do comando *cat*:

```
# cat > arquivo.txt
```

Para terminar a execução do comando, tecle "Ctrl+D" duas vezes ou "Enter" e "Ctrl+D".

Utilizando o comando *touch*

O comando *touch* é utilizado para atualizar os horários de acesso e de modificação de um arquivo existente. Caso esse arquivo não exista, será criado na data e hora especificadas no comando, cuja sintaxe é:

```
 # touch [opções] arquivo
```

Se não forem especificadas a data e a hora, o sistema utilizará a data e a hora atuais.

Também é possível criar arquivos vazios através do comando *touch*, utilizado normalmente para atualizar os horários de acesso e de modificações de um arquivo existente. Caso esse arquivo não exista, será criado na data e hora especificadas pelo comando:

```
# touch -[opções] arquivo
```

Se não forem especificadas a data e a hora, o sistema utilizará a data e hora atuais.

Exercício de fixação 5
Criando arquivos

1. Qual a diferença entre os sinais > e >> utilizados no redirecionamento?
2. Qual a diferença entre os sinais < e >?
3. Qual a diferença entre os comandos *touch* e *cat* na criação de arquivos?

Utilizando editores de texto

Podemos criar arquivos ao editar um arquivo inexistente, por intermédio de editores de texto do Linux como o vi e o Emacs.

Outra forma de criação de arquivos é por meio da utilização de editores de texto, como o vi e o Emacs. Dessa forma, ao editar um arquivo inexistente, estaremos criando esse arquivo e seu conteúdo, utilizando facilidades de edição providas pelos editores. Estudaremos detalhadamente esses editores de texto no Capítulo 5.

Criando diretórios

- Quando o usuário faz login no sistema, ele é automaticamente direcionado ao seu diretório home, onde possui permissão para criar seus arquivos e diretórios.
- A separação dos arquivos em seus diretórios home facilita aos usuários encontrarem os seus arquivos e mantê-los protegidos de outros usuários.

Para a criação de diretórios, é utilizado o comando *mkdir*.

Quando o usuário faz login em um sistema Linux, ele é automaticamente direcionado para o seu diretório home, onde tem permissão para criar arquivos e diretórios. Essa separação dos arquivos de cada usuário em seus diretórios pessoais facilita o usuário a encontrar os seus arquivos e mantê-los reservados, protegidos dos demais usuários e de outros grupos de usuários, ao qual ele não pertence. Para a criação de diretórios, é utilizado o comando *mkdir*, cuja sintaxe é:

```
# mkdir -[opções] nome ou caminho completo do diretório
```

Para criarmos uma pequena árvore de diretórios, como, por exemplo, os diretórios */home/aluno/documentos* e */home/aluno/documentos/planilhas*, podemos trabalhar de duas formas. A primeira é criando um a um os diretórios da hierarquia, como mostra o exemplo:

```
# mkdir documentos

# cd documentos (direciona para o novo diretório criado)

# mkdir planilhas
```

A segunda forma é feita utilizando um único comando. Para gerar a hierarquia dos diretórios e subdiretórios, devemos digitar o nome do diretório, incluindo os diretórios de hierarquia superior utilizando a opção *-p* do comando *mkdir*, que cria os diretórios recursivamente, como no exemplo:

```
# mkdir -p documentos/planilhas
```

Para criar um caminho na árvore de diretórios, podemos trabalhar de duas formas diferentes. Criando um a um os diretórios, como no exemplo:

```
# mkdir documentos
# cd documentos (direciona para o novo diretório criado)
# mkdir planilhas
```

Ou criando os diretórios recursivamente, utilizando um único comando:

```
# mkdir -p documentos/planilhas
```

Exercício de fixação 6
Criando diretórios

Qual a diferença entre os comandos *mkdir* e *mkdir –p*?

Copiando arquivos e diretórios

A cópia de arquivos pode ser necessária por diversos motivos, como, por exemplo, para transportar uma cópia de um arquivo para outro computador utilizando um pen drive ou para fazer cópias de segurança dos arquivos de um diretório, ação recomendável para a prevenção contra problemas que possam ocorrer no disco rígido. A cópia de arquivos pode ser feita com o comando *cp*, cuja sintaxe é:

```
# cp -[opções] origem destino
```

A origem é o caminho completo ou relativo (caso o arquivo esteja no diretório corrente) do arquivo ou dos arquivos que serão copiados e o destino é o caminho completo ou relativo do local para onde será gerada a cópia do arquivo e o nome que este terá no destino. Pode-se omitir o nome, se quisermos manter o mesmo nome da origem. Para copiar diretórios, deve ser utilizada a opção *-r* do comando *cp*, que faz cópias recursivas. A seguir são apresentados alguns exemplos de uso do comando *cp*.

```
# cp /home/aluno/documento.txt /tmp
# cp /home/aluno/documento.txt /tmp/documento2.txt
# cp -r /home/aluno /tmp
```

Se o diretório corrente for */home/aluno*, o caminho relativo pode ser utilizado:

```
# cp documento.txt /tmp
```

Se, ao contrário, estivermos localizados no diretório de destino (/tmp), o comando pode ser simplificado utilizando o caractere "." (ponto), que significa o diretório corrente:

```
# cp ../home/aluno/documento.txt .
```

O comando de cópia exige que sempre sejam definidas a origem e o destino, mesmo de forma simplificada. Caso contrário, será exibida uma mensagem de erro. Pode-se utilizar, também, os caracteres curinga "*" e "?" sem aspas. Dessa forma, todos os arquivos que atenderem à especificação múltipla serão copiados. O comando do exemplo abaixo copia todos os arquivos que têm seus nomes começando pela palavra "planilha" do diretório */home/aluno* para o diretório */tmp*.

```
# cp /home/aluno/planilha* /tmp
```

A cópia de arquivos possui diversas utilidades, como transportar arquivos entre computadores utilizando uma mídia removível, gerar cópias de segurança, entre outras. Sintaxe do comando de cópia:

```
# cp -[opções] origem destino
```

A origem e o destino devem ser obrigatoriamente informados. Caso contrário, será exibida mensagem de erro.

Exemplos de uso:

```
# cp /home/aluno/documento.txt /tmp

# cp /home/aluno/documento.txt /tmp/documento2.txt

# cp -r /home/aluno /tmp

# cp ../home/aluno/documento.txt .

# cp /home/aluno/planilha* /tmp
```

Removendo arquivos e diretórios

- O comando *rm* é utilizado para remover arquivos do sistema
- O nome do arquivo pode conter todo o caminho na estrutura hierárquica ou só o nome do arquivo, se este estiver no diretório corrente. Podem também ser utilizados caracteres curinga.
- Nem sempre será solicitada, pelo sistema, a confirmação do usuário para a execução da remoção.
- Para a remoção de diretórios, deve ser utilizada a opção *-r* do comando *rm* ou o comando *rmdir* caso o diretório esteja sem nenhum conteúdo.

O comando utilizado para remover arquivos é o *rm* e as mesmas regras vistas para o comando *cp*, se aplicam ao comando *rm*, cuja sintaxe é:

```
# rm -[opções] arquivo
```

Onde o nome do arquivo pode conter os diretórios do caminho na estrutura hierárquica até o local do arquivo, seguido do nome do arquivo. A remoção de arquivos deve ser feita com atenção, pois nem sempre será solicitada, pelo sistema, a confirmação do usuário para a execução da remoção. Da mesma forma que, para a cópia, a remoção de todos os arquivos, inclusive os diretórios, pode ser feita utilizando o comando *rm* com a opção *-r*. Diretórios sem conteúdo também podem ser removidos com o comando *rmdir*.

Exercício de fixação 7
Removendo arquivos

- O que faz o comando *rm –rif* ?
- Qual a diferença entre os comandos *rmdir* e *rm –rf* ?

Movendo arquivos e diretórios

O comando *mv* pode ser utilizado de duas formas: para mover arquivos da origem para o destino ou para renomear arquivos, trocando apenas seu nome, mantendo-o no diretório original.

Sintaxe do comando *mv*:

```
# mv [opções] origem destino
```

Exemplos de uso:

```
# mv /home/aluno/documento.txt /tmp

# mv /home/aluno/documento.txt /home/aluno/documento2.txt
```

O comando *mv* pode ser utilizado de duas formas: para mover arquivos da origem para o destino ou para renomear arquivos, trocando apenas seu nome, mantendo-o no diretório original. As mesmas regras vistas para os comandos *cp* e *rm* também se aplicam ao comando *mv*, cuja sintaxe é:

```
# mv -[opções] origem destino
```

Para mover o arquivo "documento.txt" do diretório */home/aluno* para o diretório */tmp*, devemos executar o comando:

```
# mv /home/aluno/documento.txt /tmp
```

Para renomear o arquivo "documento.txt" para "documento2.txt", devemos executar o comando:

```
# mv /home/aluno/documento.txt /home/aluno/documento2.txt
```

Exercício de fixação 8
Renomeando arquivos

O comando *mv* também serve para renomear arquivos?

Listando arquivos e diretórios

O comando *ls* é utilizado para a visualização de arquivos. As regras utilizadas nos comandos anteriores também podem ser aplicadas ao comando *ls*. Sua sintaxe é:

- `# ls [opções] arquivo`

Uma das opções mais utilizadas pelo usuário é a opção *-l*, para listar arquivos e diretórios no formato "longo", com os seguintes detalhes:

- Tipo do arquivo, permissões, número de links, dono do arquivo, grupo do arquivo, tamanho em bytes, data da última alteração e nome.
  - Os comandos *ll*, *dir* e *vdir* também podem ser utilizados com pequenas diferenças em relação ao comando *ls*.

O comando *ls* é utilizado para listar arquivos e diretórios. Sua sintaxe é:

```
# ls -[opções] [arquivo]
```

Uma das opções mais utilizadas do comando *ls* é a opção *-l*, que permite listar arquivos e diretórios no formato "longo", que exibe uma série de informações as respeito dos arquivos listados, como: permissões de acesso, número de links, dono do arquivo, grupo do arquivo, tamanho em bytes, data da última alteração e nome do arquivo. O comando *ls* também pode utilizar as mesmas regras aplicadas aos comandos anteriores.

Existem variações do comando *ls* que podem ser utilizadas com as mesmas funcionalidades. Os comandos *ll*, *dir* e *vdir* também podem ser utilizados para listar arquivos. O comando *ll* possui a mesma função do comando *ls* com as opções *-la*. O comando *dir* possui a mesma função do comando *ls* utilizado sem opções. Já o comando *vdir* possui a mesma função do comando *ls* utilizado com a opção *-l*. É importante ressaltar que as mesmas opções do comando *ls* podem ser utilizadas para os comandos *ll*, *dir* ou *vdir*.

Exercício de fixação 9
Listando arquivos

O que significam as opções –*lat* do comando *ls*?

Procurando arquivos e diretórios

O comando *find* procura o(s) arquivo(s) ou diretório(s) desejado(s) no caminho fornecido como parâmetro. Sua sintaxe é:

- `# find [caminho] [expressão]`
- O comando *locate* faz uma busca em um banco de dados que contém os arquivos criados pelo usuário e lista aqueles que satisfaçam ao padrão desejado.
- Esse comando permite acesso mais rápido a esses arquivos, contanto que não tenham sido criados depois da última atualização do banco de dados. Podemos forçar a atualização desse banco de dados por meio do comando *updatedb*.
- O comando *locate* tem sintaxe mais simples e procura o nome do arquivo isolado ou como parte de um nome, como se houvesse caracteres-curinga antes e depois do nome do arquivo. Para achar os arquivos *books* e *school*, podemos utilizar o comando:
- `# locate oo`

O comando *find* é utilizado para fazer buscas por arquivos e diretórios, fornecendo como resultado o caminho completo para o(s) arquivo(s) e diretório(s) encontrado(s). É possível, também, utilizar caracteres-curinga para ampliar a pesquisa a um conjunto de nomes que atendam a uma especificação múltipla. Formato do comando:

```
# find [caminho] [expressão]
```

Esse comando faz uma busca pela expressão definida como parâmetro, em todos os diretórios e subdiretórios especificados também como parâmetros no campo caminho, retornando os resultados da busca, caso existam. O comando *find* possui uma série de funcionalidades: busca de arquivos sem dono, busca de arquivos com o bit SUID ativado, execução de comandos sobre o resultado da busca, entre outros. Para mais detalhes a respeito do comando *find*, sua página de manual pode ser consultada.

Podemos utilizar, ainda, o comando *locate*, que faz buscas por arquivos, consultando um banco de dados que contém os arquivos criados pelo usuário. Esse comando permite acesso mais rápido a esses arquivos, contanto que não tenham sido criados depois da última atualização do banco de dados. Para forçar a atualização desse banco de dados, utilizamos o comando *updatedb*. O comando *locate* tem sintaxe mais simples e funciona como se houvesse caracteres antes e depois do nome do arquivo, isto é, procura-se o nome do arquivo isoladamente ou como parte de um nome. Por exemplo, para achar os arquivos *books* e *school*, podemos utilizar o seguinte comando:

```
# locate oo
```

Navegando pela árvore de diretórios

O diretório que fica no nível hierárquico mais alto ou no topo da árvore de diretórios é o raiz "/".

Cada caminho é representado por uma sequência única de diretórios separados pelo caractere "/" (sem aspas), que é o nome do caminho (path name). Existem dois tipos de caminhos:

- Caminhos absolutos, iniciados no diretório-raiz.
- Caminhos relativos, iniciados no diretório corrente (sem utilizar a barra inicial).
  - O Linux possui dois tipos especiais de nomes de diretórios: diretório ponto ".", que significa o diretório corrente, como vimos anteriormente, e diretório dois pontos "..", que representa o diretório um nível acima do diretório corrente. O diretório que fica num nível imediatamente acima do corrente também é chamado de diretório-pai (parent directory). Esses dois tipos especiais de diretórios são muito utilizados para a navegação na estrutura de diretórios do Linux. Se o diretório corrente é o */home/aluno*, por exemplo, podemos fazer referência ao diretório */home* somente digitando "..". Outro exemplo seria o caminho *../aluno/documentos*, que significaria o mesmo que */home/aluno/documentos*.

Existem dois tipos especiais de nomes de diretórios utilizados para a navegação na estrutura de diretórios do Linux:

- Ponto ("."), que representa o diretório corrente.
- Dois pontos (".."), que representa o diretório um nível acima do diretório corrente.

O caminho "../aluno/documentos" significa o mesmo que */home/aluno/documentos*.

O diretório que fica um nível imediatamente acima do diretório corrente é chamado de diretório-pai (parent directory).

Conforme vimos anteriormente, os diretórios do Linux são organizados hierarquicamente numa estrutura do tipo "árvore invertida". O diretório que fica no nível hierárquico mais alto ou no topo da árvore de diretórios é o diretório-raiz, representado pela barra normal ("/").

Essa forma de estrutura em árvore permite que possa haver arquivos ou diretórios com os mesmos nomes, diferenciados pelo sistema como elementos distintos devido aos caminhos diferentes (paths) percorridos na estrutura. Cada caminho é representado por uma sequência de diretórios separados pelo caractere "/" (sem aspas). Essa sequência representa o nome do caminho (path name) do arquivo ou diretório.

Existem dois tipos de caminhos: os absolutos, que se iniciam sempre no diretório-raiz, e os relativos, que são baseados no diretório corrente, em vez de se utilizar o diretório-raiz como início do caminho. Diferentemente do caminho absoluto, sempre iniciado com "/", o caminho relativo nunca começa com "/". Quando um subdiretório está muitos níveis abaixo do diretório-raiz, passa a ser trabalhoso utilizar o caminho absoluto como identificador desse diretório. É mais conveniente utilizar o caminho relativo.

Além dessas facilidades, existem os comandos *pwd* e *cd* para a navegação nos diretórios. O primeiro passo para navegar na estrutura é saber o local onde estamos. O comando *pwd* (print working directory) é utilizado para mostrar o diretório corrente, retornando o caminho completo de identificação desse diretório. O comando *pwd* é um dos mais simples do Linux, pois não tem opções nem entrada e só produz uma linha de saída. Outro comando utilizado para navegação na árvore de diretórios é o *cd*, que tem a função de mudar a localização do usuário para outro diretório. No exemplo, o comando *cd* é utilizado para navegar até o diretório */home/aluno*:

```
# cd /home/aluno
```

O comando *pwd* mostra o diretório corrente e é um dos mais simples, pois não tem opções nem entrada e só produz uma linha de saída.

O comando *cd* é utilizado para navegar pela árvore de diretórios. No exemplo abaixo, o comando *cd* muda a localização do usuário para o diretório */home/aluno*.

```
# cd /home/aluno
```

Empacotando e compactando arquivos e diretórios

O empacotamento de arquivos é um recurso utilizado para fazer cópias de segurança (backups) com mais rapidez, mas serve também para gerar pacotes de arquivos, transportá-los entre dois computadores ou enviá-los pela internet. O comando utilizado para empacotar arquivos é o *tar*, que pode empacotar um número praticamente ilimitado de arquivos, gerando como resultado um único arquivo cujo conteúdo é o conjunto de arquivos que foram empacotados. A sintaxe do comando *tar* é mostrada a seguir:

```
# tar [-][opções] arquivo lista de arquivos
```

Onde arquivo é o nome do arquivo *tar*, que será criado, e "lista de arquivos" é a lista de arquivos e diretórios que serão empacotados. O comando *tar* também é utilizado para desempacotar arquivos. As opções "-c" e "-x" são utilizadas para empacotar e desempacotar arquivos, respectivamente. Se quisermos especificar o nome do arquivo que será criado, devemos adicionalmente utilizar a opção "-f". O comando *tar* possui diversas opções, entre elas a opção "-v", que faz com que sejam listados os nomes dos arquivos que estão sendo empacotados ou desempacotados. Os exemplos a seguir são as formas mais utilizadas do comando *tar*:

```
# tar -cvf backup.tar /home/aluno
```

(agrupa todos os arquivos do diretório home do usuário aluno, empacotando-os no arquivo *backup.tar*).

```
# tar -xvf backup.tar
```

(extrai todos os arquivos contidos no arquivo *backup.tar* para o diretório corrente).

O comando *tar* é utilizado para empacotar e desempacotar arquivos. Sua sintaxe é:

```
# tar [-][opções] arquivo lista de arquivos
```

O empacotamento de arquivos pode ser utilizado para fazer cópias de segurança (backups), gerar pacotes de arquivos, transportá-los entre computadores por intermédio de mídias removíveis ou enviá-los pela internet.

Opções mais utilizadas do comando *tar*:

- **-c**: empacotar arquivos.
- **-x**: extrair os arquivos do arquivo compactado.
- **-f**: especificar o nome do arquivo.
- **-v**: listar o nome dos arquivos que estão sendo empacotados ou desempacotados.

Também é possível compactar os arquivos para transporte e armazenamento, gerando uma cópia menor. Essa facilidade é útil, por exemplo, quando precisamos enviar o arquivo pela

internet. Os comandos *gzip* e *bzip2* podem ser utilizados para compactar arquivos enquanto os comandos *gunzip* e *bunzip2* podem ser utilizados para descompactar arquivos. As sintaxes desses comandos:

```
# gzip [opções] [arquivo]

# bzip2 [opções] [arquivo]

# gunzip [opções] [arquivo]

# bunzip2 [opções] [arquivo]
```

O comando *gzip* gera um arquivo com o mesmo nome do arquivo original, incluindo a terminação *.gz*, enquanto o comando *bzip2* inclui a terminação *.bz2*.

É possível utilizar os comandos *gzip* e *bzip2* para descompactar arquivos, utilizando a opção–*d*.

Uma opção muito utilizada é empacotar e compactar arquivos simultaneamente, utilizando o comando *tar* com a opção adicional *-z* (compactação com o comando *gzip*) ou com a opção *-j* (compactação com o comando *bzip2*). Os exemplos a seguir mostram o uso do comando *tar* fazendo compactação de dados utilizando os algoritmos dos comandos *gzip* e *bzip2*:

```
# tar -zcvf backup.tar.gz /home/aluno

# tar -jcvf backup.tar.bz2 /home/aluno
```

Existem dois comandos que podem ser utilizados para compressão de arquivos (*gzip* e *bzip2*) e dois comandos que podem ser utilizados para descompressão de arquivos (*gunzip* e *bunzip2*), com os formatos:

```
# gzip [opções] [arquivo]

# bzip2 [opções] [arquivo]

# gunzip [opções] [arquivo]

# bunzip2 [opções] [arquivo]
```

Os comandos *gzip* e *bzip2* geram, como saída, arquivos com o mesmo nome do arquivo original, incluindo respectivamente as terminações *.gz* e *.bzip2*.

É possível ainda utilizar os comandos *gzip* e *bzip2* para descompactar arquivos, utilizando a opção *–d*.

Os arquivos gerados pela execução dos comandos anteriores serão gerados no diretório corrente. Se, no entanto, desejarmos transportá-los para uma fita, supondo que seu dispositivo esteja montado, deve-se especificar o caminho correspondente ao arquivo que representa o dispositivo, como mostra o exemplo:

```
# tar -czvf /dev/st0 /home/aluno
```

Onde *st0* é o dispositivo correspondente à unidade de fita.

Também é possível descompactar e desempacotar o arquivo simultaneamente, utilizando as opções *–zxvf* (compactação feita com o comando *gzip*) ou *-jxv* (compactação feita com o comando *bzip2*) do comando *tar*, como os exemplos:

```
# tar -zxvf backup.tar.gz

# tar -jxvf backup.tar.bz2
```

Para transportar o arquivo para uma fita, supondo que seu dispositivo esteja montado, deve-se especificar o caminho correspondente ao arquivo que representa o dispositivo, como mostra o exemplo:

```
# tar -czvf /dev/st0 /home/aluno
```

Uma opção muito utilizada é empacotar e compactar arquivos simultaneamente, utilizando o comando "tar" com as opções "-z"(compactação utilizando o algoritmo do comando *gzip*) ou "-j" (compactação utilizando o algoritmo do comando *bzip2*):

```
# tar -zcvf backup.tar.gz /home/aluno

# tar -jcvf backup.tar.bz2 /home/aluno
```

Exercício de fixação 10
Página de manuais

Qual dos comandos a seguir podemos utilizar para localizar o diretório no qual está armazenada a página de manual de um determinado comando no Linux?

a. which

b. w

c. slocate

d. whereis

e. man

Roteiro de Atividades 3

Atividade 3.1 – Conhecendo os arquivos

Indique as letras que representam os seguintes tipos de arquivos:

Regular – _____

Bloco – _____

Socket – _____

Pipe – _____

Arquivos – _____

Diretórios – _____

Links simbólicos – _____

Atividade 3.2 – Criando arquivos

1. Posicione-se em seu diretório de trabalho e crie os arquivos “exemplo1” e “exemplo2” utilizando comandos diferentes em cada caso. Ao criar um dos arquivos, entre com algumas linhas de dados pelo teclado.
2. Faça com que o arquivo vazio fique igual ao arquivo com conteúdo.

Atividade 3.3 – Criando diretórios e copiando arquivos

3. Crie o diretório “exemplos” em um subdiretório “temp” no seu diretório “home”. Utilize apenas um comando de criação para executar esta ação.
4. Mova os arquivos “exemplo1” e “exemplo 2” para o diretório “exemplos”, deixando uma cópia do segundo no diretório de origem.

Atividade 3.4 – Empacotando e compactando arquivos

5. Entre no diretório *temp/exemplos*. Empacote e compacte os arquivos criados anteriormente num arquivo chamado “exemplos.tar.gz”. Use um único comando para efetuar esta operação.
6. Crie um diretório chamado “exemplos2” e mova o arquivo “exemplos.tar.gz” para o diretório “exemplos2”.
7. Entre no diretório “exemplos2”, e com um único comando, descompacte e desempacote o arquivo “exemplos.tar.gz”.

Atividade 3.5 – Removendo arquivos e diretórios

1. Após o término das atividades, volte à situação original e remova todos os elementos criados.

4

Desvendando o Linux

objetivos

Utilizar o redirecionamento padrão de arquivos de entrada, saída e erro, para arquivos normais ou arquivos de dispositivos e criar funções a partir de uma sequência de comandos com a utilização de canalizações ou pipes.

conceitos

Entrada e saída padrão de dados e saída padrão de erros, pipes ou canalizações e comandos para manipulação de arquivos.

Entrada e saída padrão de dados e saída padrão de erros

- Os programas utilitários ou comandos do Linux desempenham funções simples e bem definidas.
- Os comandos obtêm dados como entrada, executam o processamento desses dados e fornecem resultados como saída.
- No Linux, a entrada padrão de dados é o teclado, a saída padrão de dados é o monitor e a saída padrão de erros também é o monitor.

Nesta sessão de aprendizagem veremos com mais detalhes alguns conceitos básicos do Linux, importantes para entendermos o funcionamento desse Sistema Operacional. Como todos os elementos no Linux são representados por arquivos, necessitamos conhecer também as formas de manipulação do conteúdo dos arquivos, assim como já vimos algumas operações no sistema de arquivos. A maior parte dos programas utilitários do Linux desempenha uma função simples e bem definida. Eles obtêm dados como entrada, executam o processamento desses dados e fornecem resultados como saída. Todo programa, quando é executado, possui uma entrada e uma saída de dados e uma saída de erros. No Linux, a entrada padrão de dados é o teclado, a saída padrão de dados é o monitor e a saída padrão de erros também é o monitor. A Figura 4.1 mostra a entrada e a saída padrão de dados e a saída padrão de erros.

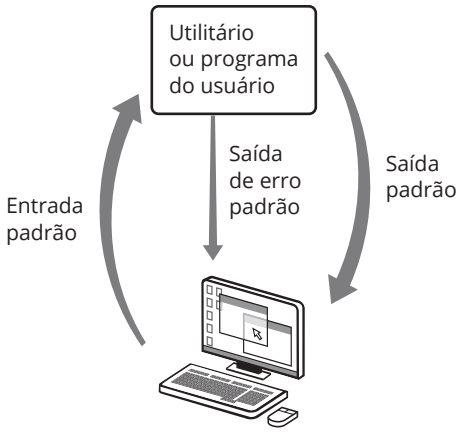


Figura 4.1
Entrada e saídas padrão.

Redirecionamento de entrada e saída

Para redirecionar arquivos padrão, devemos usar combinações de redirecionamentos e descritores de arquivos. Os descritores de arquivos só são obrigatórios no redirecionamento de erros.

Os descritores de arquivo utilizados no redirecionamento são: 0 para o arquivo de entrada padrão, 1 para o arquivo de saída padrão e 2 para o arquivo de erro padrão.

Uma das facilidades que o Linux oferece é a possibilidade de alterar a entrada e a saída padrão de dados, bem como a saída padrão de erros. Essa característica é conhecida como redirecionamento de entrada e saída. Por exemplo, suponhamos que um programa chamado "consulta" solicite o número de um cliente como entrada e apresente seus dados como saída. Normalmente, o programa receberá a entrada de dados do teclado e exibirá a saída na tela:

```
# consulta 12345

Cliente nº:12345

Nome: Aluno1

Endereço: Av. Atlântica, 110

Cidade: Rio de Janeiro

Estado: RJ
```

O Linux oferece a possibilidade de alterar a entrada e a saída padrão de dados, bem como a saída padrão de erros.

Exemplo: o programa consulta solicita o número de um cliente como entrada do teclado e apresenta seus dados como saída na tela.

- # consulta 12345
- Cliente n.: 12345
- Nome: Aluno1
- Endereço: Av. Atlântica, 110
- Cidade: Rio de Janeiro
- Estado: RJ

Se o programa de consulta tiver de ser executado para uma lista de uma centena de clientes, seria cansativo digitar os números dos clientes um a um, como entrada do programa, o que

poderia, ainda, conduzir a erros de digitação. Alternativamente, a listagem dos números dos clientes pode ser colocada em um arquivo chamado “números”, editando-o previamente, com um dos utilitários de edição de texto do Linux. Poderemos, então, redirecionar a entrada padrão do comando consulta através do símbolo “<”, que significa a entrada padrão do sistema:

```
# consulta < numeros
```

O caractere “<” instrui o interpretador de comandos (shell) a executar o programa consulta, tornando sua entrada padrão o arquivo números, em vez do teclado, isto é, redirecionando a entrada do comando. Nesse caso, apareceria na tela uma lista com todos os dados de todos os clientes, o que provavelmente encheria várias telas ocasionando perda de informações. Similarmente, em vez de utilizar a saída padrão, podemos redirecionar os dados para uma impressora utilizando o comando:

```
# consulta > /dev/lp
```

O caractere “>” instrui o shell a executar o programa consulta utilizando como saída padrão a impressora em vez do monitor de vídeo. Podemos, também, utilizar o redirecionamento de entrada juntamente com o de saída:

```
# consulta < números > clientes
```

A listagem dos números dos clientes está no arquivo “números” e a entrada-padrão do comando consulta será redirecionada através do operador de redirecionamento de entrada “<”:

```
# ./consulta < numeros
```

O caractere “>” instrui o interpretador de comandos (shell) a executar o programa consulta utilizando como saída-padrão a impressora, em vez do monitor de vídeo:

```
# ./consulta > /dev/lp
```

Podemos utilizar o redirecionamento de entrada juntamente com o de saída como no exemplo:

```
# ./consulta < números > clientes
```

A Figura 4.2 mostra como funciona o redirecionamento de entrada e saída. Os dados de entrada são lidos do arquivo números e os dados de saída são escritos no arquivo cliente. Se o arquivo cliente não existir, ele será criado com o comando do exemplo anterior. Caso ele exista, seu conteúdo será sobrescrito com os dados de saída do comando.

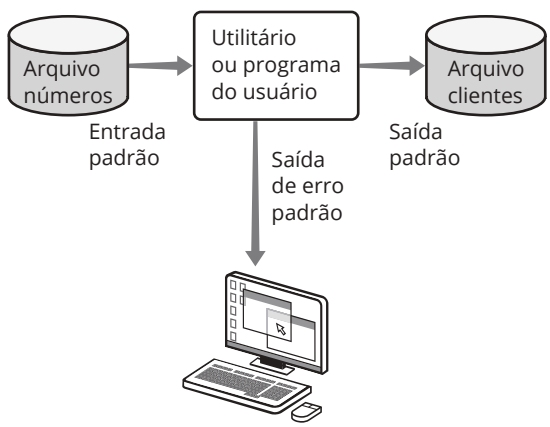


Figura 4.2
Redirecionamento de entrada e saída.

Também é possível adicionar os dados de saída no final de um arquivo, acrescentando esses dados ao conteúdo existente nesse arquivo. Para isso, devemos utilizar os caracteres ">>". Para o redirecionamento de entrada e saída de dados, podemos, opcionalmente, utilizar descritores de arquivos, mas para o redirecionamento de saída de erro, o uso de descritores de arquivos é obrigatório. Os descritores de arquivo utilizados no redirecionamento são: 0 para o arquivo de entrada padrão, 1 para o arquivo de saída padrão e 2 para o arquivo de erro padrão.

`# comando < arquivo` (redireciona a entrada padrão lendo dados do arquivo.)

`# comando 0< arquivo` (redireciona a entrada padrão lendo dados do arquivo.)

`# comando > arquivo` (redireciona a saída padrão sobrescrevendo o arquivo.)

`# comando 1> arquivo` (redireciona a saída padrão sobrescrevendo o arquivo.)

`# comando >> arquivo` (redireciona a saída padrão anexando os dados no arquivo.)

`# comando 2> arquivo` (redireciona a saída de erro padrão reescrevendo o arquivo.)

`# comando 2>> arquivo` (redireciona a saída de erro padrão anexando os dados no arquivo.)

`# comando > arquivo 2>&1` (envia a saída de erro padrão reescrevendo o arquivo.)

`# comando >> arquivo 2>&1` (envia a saída de erro padrão anexando os dados no arquivo.)

No Capítulo 3, vimos que uma das características do sistema de arquivos do Linux é a independência de dispositivo e arquivo. Essa característica, combinada com a capacidade de redirecionamento de entrada e saída, cria uma ferramenta poderosa.

Pipe ou canalização

Os utilitários do Linux desempenham funções específicas. Tarefas mais complexas são executadas pela combinação sequencial de utilitários ou comandos.

Um pipe é utilizado para executar uma sequência de comandos, passando a saída de um comando diretamente a outro comando, que a utilizará como entrada de dados.

Geralmente os utilitários do Linux são desenvolvidos para desempenhar uma única função. Tarefas mais complexas são executadas através da combinação sequencial de utilitários, com a saída de um servindo de entrada para o outro. Tal combinação é possível por meio da facilidade de canalização do Linux. Uma canalização, mais conhecida como pipe, é utilizada para direcionar a saída de um comando para outro comando, que a utilizará como entrada de dados. Essa funcionalidade é especialmente útil quando dois comandos são executados em sequência, como mostra a Figura 4.3.

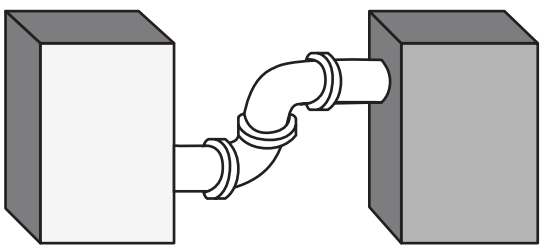


Figura 4.3
Canalização ou pipe.

Utilizando o exemplo anterior, vamos colocar os clientes em ordem numérica e mostrar suas respectivas informações por intermédio do programa consulta na tela do monitor.
Para executar estas duas funções, devemos executar os comandos:

`# sort numeros > temp` (ordena os números de identificação dos clientes em ordem crescente.)

`# ./consulta < temp` (lista os dados dos clientes, segundo a ordenação anterior.)

O mesmo resultado pode ser conseguido ao se utilizar uma canalização ou pipe, que é representado pelo caractere "|".

```
# sort números | consulta
```

Exemplo: colocar os clientes em ordem numérica e mostrar suas informações por meio do programa consulta na tela do monitor. O comando *sort* é utilizado para ordenar os números dos clientes em ordem crescente.

```
# sort numeros > temp
# ./consulta < temp
```

O mesmo resultado pode ser conseguido utilizando apenas uma linha de comando, conectando os comandos através de um pipe:

```
# sort numeros | consulta
```

A Figura 4.4 mostra um diagrama funcional da linha de comando executada no exemplo anterior. Com a funcionalidade pipe, o Linux permite que diversos comandos sejam conectados em uma sequência conhecida como canalização. Como no exemplo anterior, a saída padrão de cada comando é canalizada para a entrada do próximo comando, exceto a última que, nesse caso, é mantida como a saída padrão de forma que a listagem seja mostrada no terminal de vídeo.

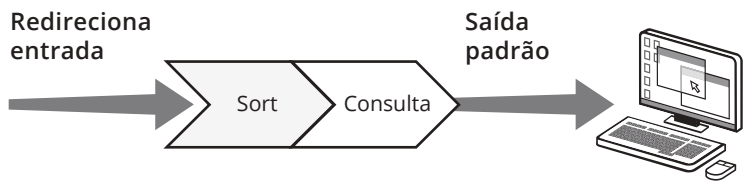


Figura 4.4 Combinação de comandos utilizando pipes.

O Linux manipula automaticamente o fluxo de dados de um programa para o outro, produzindo o mesmo efeito de uma execução de um programa que contém várias instruções. Geralmente, os arquivos de saída intermediários de uma canalização não são utilizados fora dela, pois os seus dados não são armazenados para serem utilizados por outros programas ou pelo usuário. O sistema simplesmente descarta esses dados depois que a canalização terminou de ser executada. Para preservar a saída de um arquivo intermediário no interior de uma canalização, para processamento posterior, podemos utilizar o comando *tee* como parte da canalização. O exemplo abaixo mostra como funciona o comando *tee*, que copia cada linha do arquivo de entrada para o arquivo de saída, passando todos os dados que recebe do comando *sort* para o programa consulta, mas também copia sua saída para o arquivo especificado como parâmetro, que neste exemplo é o arquivo *temp*.

```
# sort numeros | tee temp | ./consulta
```

Exercício de fixação 1
Comando *sort*

1. Qual a utilidade do comando *sort*?
2. Através da página de manual do comando, encontre a opção que organiza o arquivo com base nas colunas.
3. Pesquise três opções do comando *sort*.

Comandos para manipulação de arquivos

Neste tópico veremos alguns comandos que trabalham com arquivos. Esses comandos realizam funções de exibição, contabilização e filtragem de dados. A seguir serão apresentados em detalhes esses comandos.

Substituindo nomes de arquivos

Vimos que o comando *mv* serve para mover o arquivo da origem para o destino; na verdade uma cópia destrutiva, porque retira o arquivo de seu local original e faz uma cópia dele no local de destino, eliminando o arquivo do local de origem. O comando *mv* também pode ser utilizado para mudar o nome de arquivos, quando o usuário desejar gerar uma nova versão de um arquivo ou um novo arquivo que tenha um nome que o diferencie do anterior. É importante lembrar que o arquivo original não será mantido após ser renomeado. A sintaxe do comando *mv* é:

```
# mv [opções] arquivo novo_nome_do_arquivo
```

O comando *rename* pode ser utilizado exclusivamente para renomear arquivos, mas com a funcionalidade adicional de trocar partes de nomes, sendo muito útil quando utilizado com caracteres-curinga. No exemplo abaixo, as terminações de todos os arquivos do diretório corrente são trocadas de *.htm* para *.html*.

```
# rename .htm .html *.htm
```

O comando *mv* move o arquivo da origem para o destino, fazendo uma cópia no local de destino e eliminando o arquivo do local de origem. O comando *mv* também pode ser utilizado para mudar o nome de arquivos. É importante lembrar que o arquivo original não será mantido após ser renomeado.

A forma do comando *mv* é:

```
# mv -[opções] arquivo_origem arquivo_destino
```

O comando *rename* pode ser utilizado exclusivamente para trocar os nomes de arquivos, mas com a funcionalidade adicional de trocar parte de nome do nome dos arquivos. O comando abaixo pode ser utilizado parar trocar as terminações de todos os arquivos do diretório corrente de *.htm* para *.html*.

```
# rename .htm .html *.htm
```

Visualizando o conteúdo de arquivos

Vimos que o comando *cat* pode ser utilizado para criar um arquivo, mesmo sendo sua função principal a de concatenar arquivos. Como a sua saída padrão é a tela do monitor, ao executar o comando a seguir, o conteúdo do arquivo será mostrado na tela:

```
# cat arquivo
```

O resultado da concatenação será o próprio arquivo, pois não há nenhum outro arquivo especificado no comando para ser concatenado. O problema do comando *cat* é que, se o tamanho do texto contido no arquivo ultrapassar o número de linhas da tela do monitor, ele continuará mostrando o arquivo até o final, sem pausas, de forma que a tela só pare de rolar quando todo o conteúdo do arquivo for exibido. Logo, só serão mostradas na tela as últimas linhas do arquivo, e não veremos as linhas anteriores, ou melhor, elas serão mostradas rapidamente na tela, não havendo tempo reservado para a sua leitura. Para evitar o problema de visualização de arquivos maiores que o número de linhas da tela, devemos utilizar os comandos *more* ou *less*, que listam o arquivo dando uma pausa na listagem quando ela preenche toda a tela, permitindo que se controle a exibição do conteúdo. A sintaxe desses comandos pode ser vista abaixo:

```
# more [opções] [-num] [+/padrão] [+número_da_linha] [arquivo]

# less [opções] [arquivo]
```

Utilizando o comando *more*, se o conteúdo do arquivo for maior que a quantidade de linhas do monitor, será exibida no final da tela a expressão "--more--(x%)", onde x representa a porcentagem do arquivo que já foi exibida, indicando que há mais texto a ser visualizado. Se pressionarmos a tecla *Enter*, o comando *more* mostrará mais uma linha do texto rolando a primeira linha para cima na tela. Se quisermos mostrar mais uma tela inteira com as linhas seguintes do texto, devemos pressionar a barra de espaço. Podemos, dessa forma, controlar a exibição do texto, passando-o linha a linha na tela ou paginando-o tela a tela. Para sair do comando *more* antes de chegar ao fim do arquivo, podemos utilizar as teclas *Ctrl+C* ou a tecla *q*.

O comando *less* possui praticamente as mesmas funcionalidades do comando *more*, só que mais recursos.

O comando *cat* tem a função principal de concatenar arquivos, mas pode ser utilizado para mostrar o conteúdo de um arquivo:

```
# cat [opções] [arquivo]
```

O comando *cat* não controla a exibição na tela, deixando-a rolar até o final do arquivo e dificultando a sua visualização.

Devemos utilizar os comandos *more* ou *less*, que dão uma pausa na listagem quando ela preenche toda a tela.

```
# more [opções] [+/padrão] [+numero_da_linha] [arquivo]

# less [opções] [arquivo]
```

Exercício de fixação 2

Visualizando conteúdo de arquivos

Quais recursos o comando *less* possui a mais que o comando *more*?

Contabilizando o conteúdo de arquivos

Podemos contabilizar os elementos contidos em um arquivo texto, que são os caracteres, as palavras e as linhas de texto desses arquivos. Para isso, existe um utilitário chamado wc, que conta e imprime o número de caracteres, de palavras ou de linhas de um arquivo. A sintaxe do comando é:

```
# wc [opções] arquivo
```

Caso não seja especificado nenhum arquivo, o comando *wc* lê a entrada padrão. O comando do exemplo a seguir lista o conteúdo do diretório corrente e direciona a sua saída para a entrada do comando *wc* que, com a opção *-l*, conta o número de linhas listadas pelo comando *ls –l* e fornece, como resultado, o número de arquivos e diretórios contidos no diretório */etc*.

```
# ls -l /etc | wc -l
```

Como a primeira linha da saída do comando *ls –l* não é um arquivo ou diretório, devemos descontá-la do resultado. Veremos, nos próximos itens, comandos mais complexos, que permitem obter resultados exatos para a contagem de conteúdos. O comando *wc* também pode ser utilizado para determinar que dois arquivos não são idênticos, mostrando que não têm o mesmo número de linhas, palavras ou caracteres. No entanto, se tiverem números idênticos de linhas, palavras e caracteres, existe uma grande probabilidade de que sejam idênticos, mas não há garantia total.

O comando *wc* conta e imprime o número de caracteres, de palavras ou de linhas de um arquivo:

```
# wc [opções] [arquivo]
```

Opções do comando *wc*:

- **-l**: conta o número de linhas do arquivo.
- **-w**: conta o número de palavras do arquivo.
- **-c**: conta o número de caracteres do arquivo.

Caso não seja especificado nenhum arquivo, o comando *wc* lê a entrada padrão. Se não for escolhida uma opção, o comando *wc* conta e retorna todas as opções anteriores.

Exibindo o conteúdo inicial e final de arquivos

Para exibir o conteúdo inicial e final de arquivos texto, existem dois comandos: o *head* e o *tail*, respectivamente. O comando *head* permite que visualizemos as primeiras linhas de um arquivo. Sua sintaxe é:

```
# head [opções] [arquivo1 arquivo2 ...]
```

Caso não seja especificado nenhum arquivo, o comando *head* lê a entrada padrão. Caso seja especificado mais de um arquivo, cada um deles terá suas 10 primeiras linhas exibidas na tela e seu conteúdo será precedido pelo nome do arquivo como mostra o exemplo:

```
==> arquivo <==
```

A principal opção desse comando é a que especifica o número de linhas iniciais que serão listadas, como mostra o exemplo abaixo, que exibe as cinco primeiras linhas do arquivo *relatório.txt*:

```
# head -5 relatorio.txt
```

Se não for informado o número de linhas, o comando retornará as 10 primeiras linhas por padrão. O inverso do comando *head* é o comando *tail*, que mostra as últimas linhas de um arquivo. Sintaxe do comando *tail*:

```
# tail [opções] [arquivo1 arquivo2 ...]
```

Por exemplo, se quisermos exibir as últimas 15 linhas do arquivo *relatório.txt*, devemos digitar o comando:

```
# tail -15 relatorio.txt
```

Se não for especificado o número de linhas a serem exibidas, o comando retornará as últimas 10 linhas, por padrão.

- Para exibir o início e o final do conteúdo de um arquivo, são utilizados os comandos *head* e *tail*, respectivamente.
- O comando *head* permite que visualizemos as primeiras linhas de um arquivo. O comando do exemplo abaixo exibe as cinco primeiras linhas do arquivo *relatório.txt*.

  ```
  # head -5 relatorio.txt
  ```

- O comando *tail* mostra as últimas linhas de um arquivo. O comando do exemplo abaixo exibe as últimas quinze linhas do arquivo *relatório.txt*.

  ```
  # tail -15 relatorio.txt
  ```

- Se não forem especificados os números de linhas a serem exibidas, os comandos *head* e *tail* retornarão as primeiras ou últimas dez linhas, respectivamente.

Exercício de fixação 3
Exibindo o conteúdo de arquivos

Que saídas serão mostradas com os comandos abaixo?

- Ls -la .
- Ls – l ..
- Ls –l /etc/resolv.conf | wc –l
- Ls /etc | wc -c

Selecionando trechos de arquivos

A pesquisa e seleção de conteúdos de arquivos no Linux é uma operação muito comum, e pode ser utilizada para encontrar uma determinada informação. Vamos supor que tenhamos de encontrar no arquivo de clientes do primeiro exemplo todos os clientes que residam no mesmo estado, na mesma cidade ou na mesma rua. Existem vários utilitários do Linux que realizam a pesquisa de conteúdo, embora alguns sejam muito sofisticados para o escopo deste curso introdutório. Estudaremos o mais utilizado e que atende a maior parte das necessidades comuns do usuário. O comando *grep* é um utilitário de seleção e pesquisa de arquivos. A Figura 4.5 mostra o funcionamento do comando *grep*.

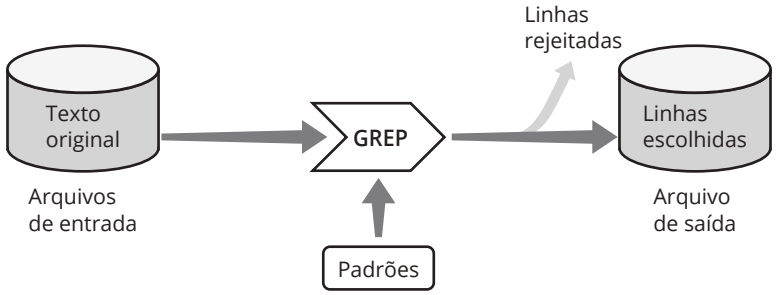


Figura 4.5
Funcionamento do comando ***grep***.

O usuário especifica um padrão que será utilizado pelo comando *grep* para fazer a pesquisa nos arquivos de entrada. Esse utilitário examina as linhas do arquivo, verificando se cada uma contém o padrão especificado. Quando o padrão é encontrado, a linha é copiada para o arquivo de saída, ou para a tela do terminal, se for utilizada a saída padrão. Se a linha não contém o padrão, é rejeitada, isto é, não é copiada no arquivo de saída.
Quando o comando *grep* termina de pesquisar os arquivos de entrada, o arquivo de saída vai conter todas as linhas que contêm os padrões desejados. A sintaxe do comando *grep* é:

```
# grep [opções] padrão [arquivo]
```

Caso não seja especificado nenhum arquivo, o comando *grep* lê a entrada padrão (teclado). Um exemplo de uso do comando *grep* pode ser conferido a seguir, onde é utilizado para selecionar somente os clientes que moram no Distrito Federal:

```
# cat clientes
12341 Aluno1 Rua Goiás, 25 Planaltina DF
12342 Aluno2 Av. Beira Rio, 1005 Vitória ES
12343 Aluno3 Rua Guarani Campinas SP
12344 Aluno4 SQS 205 F 306 Brasília DF
12345 Aluno5 Av. Atlântica, 110 Rio de Janeiro RJ
12346 Aluno6 Av. Castro Alves, 320 Salvador BA
12347 Aluno7 Rua da Praia, 215 Porto Alegre RS
12348 Aluno8 Rua das Laranjeiras,1020 Rio de Janeiro RJ
# grep DF clientes
12341 Aluno1 Rua Goiás, 25 Planaltina DF
12344 Aluno4 SQS 205 F 306 Brasília DF
```

Saiba mais

O utilitário ***grep*** também funciona como um filtro de canalização. Quando utilizado em uma canalização, seleciona algumas saídas do programa ou comando anterior e envia o resultado para a entrada do programa seguinte que irá processá-las.

Os padrões utilizados pelo comando *grep* são chamados de expressões regulares. Podemos utilizar caracteres especiais que dão maior flexibilidade ao comando *grep*:

- O comando *grep* é utilizado para seleção e pesquisa de arquivos, por meio de um padrão especificado pelo usuário.
- Quando o padrão é encontrado, a linha é copiada para o arquivo de saída. Se a linha não contém o padrão, é rejeitada.

- Quando o comando *grep* terminar de pesquisar os arquivos de entrada, o arquivo de saída conterá todas as linhas que contêm os padrões desejados:

```
# grep [opções] padrão [arquivo]
```

  - Se não for especificado nenhum arquivo, o comando *grep* lê a entrada padrão.
- O utilitário *grep* funciona também como um filtro de canalização, selecionando algumas saídas do comando anterior e enviando o resultado para a entrada do comando seguinte.
- Podemos usar caracteres especiais que dão maior flexibilidade ao comando *grep*.
- O caractere ponto (".") é um caractere especial que permite encontrar padrões com qualquer caractere na posição determinada pelo ponto.
- Caracteres entre colchetes [xyz] permitem achar o conjunto especificado de caracteres na posição determinada pelos colchetes.
- O caractere asterisco ("*") permite achar qualquer número (inclusive zero) de ocorrências do caractere que o precede. Por exemplo, a expressão "grep abc*" pode selecionar linhas que contêm "ab", "abc", "abcc", "abccc", "abcccc" etc.
- O caractere "^" seleciona as linhas que começam com o padrão sucedido por ele. Por exemplo, a expressão "grep ^Linux" seleciona as linhas de um arquivo que começam com a palavra Linux.
- O caractere "$" seleciona as linhas que terminam com o padrão precedido por ele. Por exemplo, a expressão "grep Linux$" seleciona as linhas de um arquivo que terminam com a palavra Linux.

O comando *egrep* (extended grep), como o próprio nome diz, aceita expressões regulares mais elaboradas do que o comando *grep*. Podemos utilizar os seguintes caracteres especiais adicionais:

- Números entre chaves {n,m}, que permitem encontrar padrões com n a m ocorrências do caractere que precede as chaves. Por exemplo, a expressão egrep "abc{0,2}" arquivo pode selecionar "ab", "abc" ou "abcc". Deve-se utilizar sempre aspas simples ou duplas na expressão.
- O caractere "+" permite encontrar uma ou mais letras entre os colchetes que o precedem, concatenando-as com os caracteres que o sucedem. Por exemplo, a expressão egrep "[a-z]+ove" permite achar move, aprove, love etc.
- O caractere "?" permite encontrar zero ou uma ocorrência da letra que o precede. Por exemplo, a expressão "lo?ve" permite encontrar love ou lve.
- O caractere "|" permite encontrar expressões iguais às que o antecedem e às que o sucedem. Por exemplo, a expressão egrep "love|hate" permite encontrar expressões que contêm as palavras love ou hate.
- Caracteres entre parênteses (xyz), como por exemplo egrep "love(able|ly), permitem encontrar as palavras lovable ou lovely.

Existe ainda o comando *rgrep*, que permite estender a pesquisa de um padrão de caracteres recursivamente, ou seja, no diretório corrente e em seus subdiretórios, e exibe não só a linha procurada como também o nome do arquivo que contém a linha. Utilizando as opções *–r* e *–n* no comando *grep*, obtém-se o mesmo resultado.

Outra alternativa ao comando *grep* é o comando *fgrep* (fixed grep ou fast grep), que funciona similarmente ao comando *grep*, mas não reconhece nenhum caractere especial em expressões regulares. Todos os caracteres representam apenas a si mesmos, não havendo significados metafóricos.

Outros comandos alternativos ao *grep*:

- O comando *egrep* (extended grep) permite usar caracteres especiais adicionais aos caracteres do comando *grep*.
- O comando *rgrep* permite estender a pesquisa de um padrão de caracteres recursivamente, ou seja, no diretório corrente e em seus subdiretórios, exibindo não só a linha procurada como também o nome do arquivo que contém a linha. A opção *–r*, do comando *grep*, possui a mesma funcionalidade do comando *rgrep*.
- O comando *fgrep* (fixed grep ou fast grep) funciona similarmente ao comando *grep*, mas não reconhece nenhum caractere especial em expressões regulares. Todos os caracteres representam apenas eles mesmos, não havendo significados metafóricos.

Comparação entre arquivos

O comando *cmp* compara os arquivos e mostra a posição em que aparece a primeira diferença.

O exemplo a seguir compara dois arquivos que têm um caractere diferente na posição 17 da linha 10:

```
# cmp arquivo1 arquivo2
 arquivo1 arquivo2 differ: char 17, line 10
```

Podemos comparar o conteúdo de arquivos e determinar quais são as suas diferenças, se não possuírem conteúdos idênticos. Uma necessidade de comparação surge quando o usuário faz uma cópia do arquivo e quer verificar se essa cópia foi bem-sucedida. Um dos comandos utilizados para a comparação entre dois arquivos é o *cmp*, que compara os arquivos e mostra a posição em que aparece a primeira diferença. A Figura 4.6 mostra como é o funcionamento do comando *cmp*.

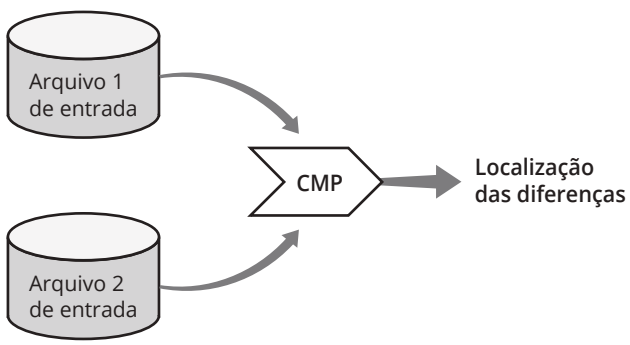


Figura 4.6
Funcionamento do comando *cmp*.

Por exemplo, se compararmos dois arquivos que possuem o 17º caractere diferente, estando este localizado na linha 10, teremos a resposta:

```
# cmp arquivo1 arquivo2
 arquivo1 arquivo2 differ: byte 17, line 10
```

Outro comando utilizado para comparar arquivos é o *diff*, que exibe as diferenças entre eles, gerando uma lista de procedimentos que podem ser aplicados ao arquivo original, usando o utilitário patch, para tornar os arquivos idênticos. O comando *diff* possui mais recursos que o comando *cmp*. A saída do comando *diff* é uma lista de ações a serem editadas que, quando aplicadas ao conteúdo do primeiro arquivo, geram o arquivo editado. As ações a serem editadas incluem adição de uma linha, eliminação de uma linha ou, então, mudança em uma linha.

- O comando *diff* gera ações que incluem adição de uma linha, eliminação de uma linha ou mudança de uma linha.
- O comando *patch* é utilizado para ler a lista de ações geradas pelo *diff* e fazer as devidas alterações no arquivo original para deixá-lo igual ao arquivo editado.
- O comando *diff* pode ser muito eficiente para guardar diferentes versões do mesmo arquivo, armazenando apenas as diferenças entre eles e, com isso, economizando espaço em disco.

Ordenação em arquivos

O comando *sort* obtém as linhas de um ou mais arquivos de entrada e produz um arquivo de saída com as linhas em ordem classificada. Sintaxe do comando *sort*:

```
# sort -[opções] [arquivo]
```

Cada linha de um arquivo de entrada é tratada como campos, separados uns dos outros por espaços, como se fossem colunas.

O conteúdo de um arquivo pode ser ordenado antes de ser processado. O comando *sort* obtém as linhas de um ou mais arquivos de entrada e a seguir as processa, produzindo um arquivo de saída que contém as linhas em ordem classificada. Sintaxe do comando *sort*:

```
# sort [opções] [arquivo]
```

O comando *sort* trata cada linha de um arquivo de entrada como uma série de um ou mais campos, separados uns dos outros por espaços, como se fossem colunas. O exemplo a seguir lista o arquivo de entrada "clientes" com o primeiro campo em ordem numérica decrescente.

```
# sort -nr clientes

12348 Aluno8 Rua das Laranjeiras,1020 Rio de Janeiro RJ

12347 Aluno7 Rua da Praia, 215 Porto Alegre RS

12346 Aluno6 Av. Castro Alves, 320 Salvador BA

12345 Aluno5 Av. Atlântica, 110 Rio de Janeiro RJ

12344 Aluno4 SQS 205 F 306 Brasília DF

12343 Aluno3 Rua Guarani, 510 Campinas SP

12342 Aluno2 Av. Beira Rio, 1005 Vitória ES

12341 Aluno1 Rua Goiás, 25 Planaltina DF
```

Se quisermos ordenar o arquivo por estado, deveremos utilizar o comando a seguir, que ordena alfabeticamente pelo número da coluna especificada na opção *-k*:

```
# sort -k5 clientes
12345 Aluno5 Av. Atlântica, 110 Rio de Janeiro RJ
12341 Aluno1 Rua Goiás, 25 Planaltina DF
12343 Aluno3 Rua Guarani, 510 Campinas SP
12346 Aluno6 Av. Castro Alves, 320 Salvador BA
12344 Aluno4 SQS 205 F 306 Brasília DF
12348 Aluno8 Rua das Laranjeiras,1020 Rio de Janeiro RJ
12347 Aluno7 Rua da Praia, 215 Porto Alegre RS
12342 Aluno2 Av. Beira Rio, 1005 Vitória ES
```

Roteiro de Atividades 4

Atividade 4.1 – Pesquisando em arquivos

1. Copie ou crie o arquivo *atividade*, contendo as seguintes linhas:

```
abcdefghijklmUnixnopqrstuvwxyz
abcdefghiUnixjklmnopqrstuvwxyzUnix
abcdefghijklmnopqrstuvwxyzLinux
abcdefghijklmnopqrstuUnixvwxyz
Unixabcdefghijklmnopqrstuvwxyz
abcdefghijklmnopqrstuvwxyzUnix
abcdefghijklmnopLinuxqrstuvwxyz
abcdefghUnixijklmnopqrstuUnixvwxyz
```

Supondo que você não conheça o conteúdo do arquivo, mas imagina que deveria haver a palavra Unix em todas as suas linhas, utilize uma linha de comando para mostrar na tela as linhas, com seus respectivos números, nas quais está faltando a palavra Unix; salve esse resultado no arquivo chamado *atividade_temp*.

Atividade 4.2 – Contabilizando arquivos

Utilizando o comando *grep,* encontre mais de uma forma de mostrar na tela o número de linhas em que a palavra Unix aparece pelo menos uma vez no arquivo *atividade* (não mostrar as linhas).

Atividade 4.3 – Controlando a exibição do conteúdo de arquivos

Utilizando uma canalização, liste na tela do monitor a terceira e a quarta linhas do arquivo *atividade*.

Atividade 4.4 – Combinando comandos para criar novas funcionalidades

Utilizando o arquivo *clientes*, crie um comando utilizando os conceitos de canalização e redirecionamento que execute passo a passo a mesma função do comando abaixo:

```
# sort -n clientes
```

12348	Aluno8	R. das Laranjeiras, 1.020	Rio de Janeiro	RJ
12347	Aluno7	Rua da Praia, 215	Porto Alegre	RS
12346	Aluno6	Av. Castro Alves, 320	Salvador	BA
12345	Aluno5	Av. Atlântica, 110	Rio de Janeiro	RJ
12344	Aluno4	SQS 205 F 306	Brasília	DF
12343	Aluno3	Rua Guarani	Campinas	SP

12342	Aluno2	Av. Beira Rio, 1.005	Vitória	ES
12341	Aluno1	Rua Goiás, 25	Planaltina	DF

Passo a passo significa que, a cada execução do comando, uma linha será colocada em ordem. Após cada comando, verifique a ordenação no arquivo *clientes*.

5

Edição de texto

objetivos

Conhecer as formas de entrar e sair do editor Vi, bem como seus comandos de movimentação, inserção e remoção de texto; salvar as modificações e procurar por padrões de caracteres e palavras no texto.

conceitos

Processadores de texto e editores de texto.

Processadores de texto

Os processadores de texto oferecem suporte à criação, edição e formatação de documentos que suportam inclusão de textos, tabelas, equações científicas, entre outros, utilizando um conjunto de funcionalidades próprias. O Linux oferece um conjunto de ferramentas para processamento de texto, com destaque para o Writer, do pacote Open Office.

As primeiras aplicações do Unix foram as de criação de documentos de texto e de desenvolvimento de software, que exigiram o desenvolvimento de ferramentas eficientes de processamento de texto. As ferramentas de processamento de texto do Linux são uma família de utilitários que atuam em todas as fases de preparação de um documento.

Nesta sessão de aprendizagem, não abordaremos o uso de processadores de texto, destacando apenas alguns editores de texto mais utilizados do Linux, com destaque para o editor Vi, que será visto em detalhes.

Editores de texto

Os editores de texto são utilizados para a edição de arquivos que contenham somente texto, sem qualquer tipo de formatação. Eles permitem a entrada, remoção ou substituição de texto, além de oferecer diversas funcionalidades: busca, correção de erros, cópia e colagem de texto, entre outros. Os editores de texto podem ser divididos em duas categorias: os editores de tela que exibem o texto na tela inteira e os editores de linha que exibem o conteúdo do arquivo linha a linha. Esta sessão de aprendizagem será baseada no editor de textos Vi, que será visto no próximo tópico. Por isso, não entraremos em detalhes no modo de utilização dos outros editores apresentados a seguir.

- **Emacs**: um editor de texto de tela cheia que foi desenvolvido por Richard Stallman, guru e fundador da Free Software Foundation. O Emacs é um poderoso ambiente de edição de textos, que inclui funções adicionais chamadas de extensões. Entre essas funções estão

o envio de e-mails de dentro do editor, a conversão de dados de arquivos em outros formatos e até mesmo a compilação de programas editados pelo Emacs. Para abrirmos um arquivo utilizando o editor Emacs, devemos utilizar o comando *emacs*, seguido do nome do arquivo. Se o arquivo especificado na linha de comando não existir, ele será criado. Caso contrário, o arquivo existente ficará disponível para edição. Se o Emacs for aberto em um ambiente gráfico, como o GNOME ou o KDE, será aberta uma janela com uma interface baseada em menus. Para ativar uma das opções do menu, basta posicionar o cursor do mouse e clicar sobre a opção. Serão exibidas então as opções de cada menu.

- **Pico**: o Pine Composer é um editor de texto de tela cheia desenvolvido pela Universidade de Washington, que é parte integrante do programa cliente de e-mail Pine. Apesar disso, o Pico pode ser utilizado para editar arquivos texto independentemente do Pine. O Pico possui várias funcionalidades, como busca de texto e verificador de erros, entre outras. Para abrirmos um arquivo utilizando o editor Pico, devemos utilizar o comando *pico*, seguido do nome do arquivo. Se o arquivo especificado na linha de comando não existir, ele será criado. Caso contrário, o arquivo existente ficará disponível para edição.
- **JOE**: o Joe's Own Editor é um editor de texto de tela cheia criado por Joseph H. Allen em 1991 e distribuído pela GPL. O JOE possui uma série de funcionalidades: um sistema integrado de ajuda, busca de texto, emulação de funcionalidades de outros editores, entre outras. Para abrirmos um arquivo utilizando o editor JOE, devemos utilizar o comando *joe*, seguido do nome do arquivo. Se o arquivo especificado na linha de comando não existir, ele será criado. Caso contrário, o arquivo existente ficará disponível para edição.
- **Ed**: um editor de texto de linha criado em 1971, pelo criador do Unix, Ken Thompson. Para abrirmos um arquivo utilizando o editor Ed, devemos utilizar o comando *ed*, seguido do nome do arquivo. Se o arquivo especificado na linha de comando não existir, ele será criado. Caso contrário, o arquivo existente ficará disponível para edição. Quando um arquivo é aberto pelo Ed, por padrão, a última linha do arquivo é exibida, permitindo a inserção de texto após ela. Os comandos do Ed só afetam a linha que está sendo exibida na tela, a não ser que outra linha, ou mesmo um conjunto de linhas, seja especificado.

Saiba mais sobre os editores de texto Emacs (http://www.gnu.org/software/emacs), Pico (http://www.washington.edu/pine/man/#pico), Joe (http://joe-editor.sourceforge.net) e Ed (http://www.gnu.org/software/ed/ed.html).

Editor Vi

Visual Editor (Vi) é um dos mais antigos editores de texto para sistemas Unix-like. Desenvolvido em Berkeley, passou a ser incorporado por padrão às diversas versões do sistema operacional Unix. O Vi possui uma interface bastante simples, sem os requintados recursos de edição oferecidos pelos processadores de textos para ambientes gráficos. Apesar de sua simplicidade, o Vi oferece vários recursos de edição de texto, dos simples aos mais sofisticados e é largamente utilizado por administradores de sistema e programadores. Por esse motivo, será o editor que detalharemos nesta sessão de aprendizagem.

Saiba mais

O buffer nada mais é do que um local na memória RAM para onde é copiado o conteúdo de um arquivo quando este está sendo editado.

O Vi copia o conteúdo do arquivo de texto que está sendo editado para um buffer de edição na memória principal, utilizando a tela do monitor de vídeo como uma janela para visualização deste buffer. A tela pode ser movimentada para cima ou para baixo mostrando outras linhas do texto armazenado no buffer, possibilitando ao usuário a navegar por todo o arquivo.

Na Figura 5.1 vemos como a tela do terminal mostra o conteúdo do buffer. Para editar o texto, o usuário simplesmente move o cursor pela tela, utilizando comandos de um só caractere ou utilizando as setas do teclado. Outros comandos de edição de texto são utilizados para inserir, modificar e remover texto. O comando especifica se a alteração a ser executada é em um caractere, em uma palavra ou em uma linha. Os resultados de cada comando são refletidos imediatamente no vídeo.

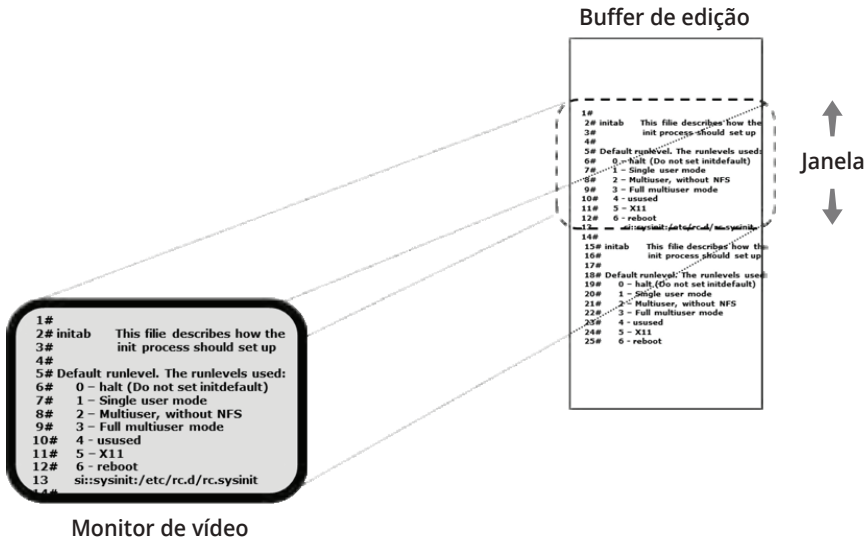


Figura 5.1
Exibição do conteúdo do buffer.

Os comandos do Vi permitem fazer mudanças em massa no documento inteiro ou em partes dele. As alterações realizadas são refletidas imediatamente no texto após a execução de um comando.

O Vi copia o conteúdo de um arquivo de texto para o buffer de edição na memória principal e utiliza a tela do monitor de vídeo como uma janela para a visualização do buffer. A tela pode ser movimentada para cima ou para baixo mostrando as demais linhas do texto.

Quando o Vi salva as alterações realizadas, ele copia o texto editado no buffer para o arquivo armazenado de forma permanente, substituindo o conteúdo anterior. O comando *vi* é utilizado para executar o editor Vi e sua sintaxe é:

```
# vi [opções] [arquivo]
```

Se o nome do arquivo for omitido, o Vi abrirá um novo arquivo sem nome no buffer de edição, que ficará vazio, aguardando o usuário digitar seu texto. Os caracteres "~" na coluna da esquerda da tela indicam que não há texto no arquivo, nem mesmo espaços em branco. Na última linha da tela é mostrado o nome do arquivo e informado se ele é novo ou se já existia. Caso o arquivo já exista, é mostrado o seu nome, o número de linhas e o número de caracteres que ele possui. Para sair do Vi sem salvar o arquivo editado, deve-se digitar a tecla *Esc* para sair do modo de inserção, digitar o comando *:q!* e em seguida teclar *Enter*. Dessa forma, a última versão do arquivo é preservada e as alterações feitas são descartadas. O Vi, por padrão, não permite a saída do editor com o descarte das alterações feitas no buffer. O ponto de exclamação após o comando *:q* permite sair do Vi, mesmo que o buffer tenha sido modificado.

O nome do arquivo deve ser único dentro do seu diretório. Um nome de arquivo pode incluir qualquer caractere, exceto alguns caracteres especiais como "/", "*" e "&", entre outros. É possível incluir espaços em um nome de arquivo digitando uma barra invertida ("\") antes do espaço.

Durante a edição de um texto, é possível sair do editor a qualquer momento, salvando as modificações realizadas e retornando ao prompt do Linux. O comando para sair do editor, salvando as modificações realizadas, é *ZZ* (em maiúsculas). Também é possível sair do editor, salvando as modificações feitas, utilizando os comandos *w* e *q*, simultaneamente (*:wq*).
O comando *:w* é utilizado para salvar as modificações sem sair do Vi e o comando *:q* é utilizado para sair do editor desde que não tenha sido feita nenhuma modificação.

O comando *vi* inicia o editor Vi e sua sintaxe é:

```
# vi [opções] [arquivo]
```

Para fechar o arquivo, existem algumas opções:

- **ZZ**: salva as alterações e fecha o arquivo.
- **:wq**: salva as alterações e fecha o arquivo.
- **:q!**: sair do arquivo descartando as alterações feitas.

Modos do editor Vi

Existem três formas básicas ou modos de trabalhar com o editor Vi.

- **Modo comando**: quando se abre um arquivo para edição, o Vi entra no modo de comando. Os comandos realizam diversas funções, como rolar através do texto, fazer buscas por trechos do texto, apagar caracteres, palavras ou linhas, entre outros. Para facilitar a execução de comandos no Vi, pode-se ativar a exibição dos números das linhas do arquivo, através do comando *:set number*.
- **Modo de inserção**: esse modo é utilizado quando se deseja inserir texto em um arquivo. A mudança do modo de comando para o modo de inserção é feita por intermédio de uma das opções:
  - **i**: entra no modo de inserção de texto, começando na posição atual do cursor.
  - **a**: entra no modo de inserção de texto, começando um caractere depois da posição do cursor.
  - **A**: entra no modo de inserção de texto, começando no fim da linha onde está o cursor.
  - **o**: entra no modo de inserção de texto, começando uma linha abaixo da posição atual do cursor.
  - **O**: entra no modo de inserção de texto, começando na posição atual do cursor e passando a linha atual uma linha para baixo.
  - **s:** entra no modo de inserção de texto, substituindo o caractere da posição atual do cursor.
  - **cw**: entra no modo de inserção de texto, substituindo uma palavra a partir do caractere onde o cursor está posicionado. Outras variações do comando *c* também podem ser utilizadas para entrar no modo de edição. Essas opções de comando serão vistas no tópico “Substituindo texto”.

Para sair do modo de inserção, basta teclar *Esc*.

- **Modo de execução**: esse modo é uma variante do modo de comandos do Vi, que também permite a execução de comandos do Linux sem precisar sair do editor Vi. Os comandos do modo de execução sempre começam com dois pontos (“:”). Vários comandos de edição de arquivos do Vi também podem ser executados nesse modo. Para executar comandos do Linux, deve-se digitar no prompt do Vi os caracteres *:!*, seguidos pelo comando. Por exemplo, para conhecer a localização do arquivo que está sendo editado, deve-se executar o seguinte comando:

```
:!pwd
```

- **Modo de comando**: quando se abre um arquivo, o editor entra no modo de comando, que permite realizar funções como rolar através do texto, buscar por palavras, apagar caracteres específicos, palavras ou linhas de texto, entre outras.
- **Modo de inserção**: a mudança do modo de comando para o modo de inserção é feita por meio dos comandos *i*, *a*, *A*, *o*, *A*, *O*, *s* ou *cw* no modo de comando. Esse modo é utilizado quando se deseja inserir texto em um arquivo. Para retornar ao modo de comando, basta teclar "Esc".
- **Modo de execução**: este modo também permite a execução de comandos do Vi. Além disso, permite a execução de comandos do Linux dentro do editor Vi.

Movimentando o cursor

Teclas de movimentação do cursor:

- **h**: move o cursor uma posição à esquerda.
- **j**: move o cursor uma linha abaixo.
- **k**: move o cursor uma linha acima.
- **l**: move o cursor uma posição à direita.

As setas também podem ser utilizadas para movimentar o cursor.

No modo de comando, é possível posicionar o cursor em qualquer lugar do arquivo, o que é necessário para fazer as modificações, remoções ou cópias de texto, a partir da posição do cursor. Existem comandos no Vi para mover o cursor, que permitem subir, descer, ir para a esquerda ou para a direita, caractere por caractere; ir para frente ou para trás de bloco em bloco de texto, como palavras, frases ou parágrafos; ir para frente ou para trás de tela em tela, em um arquivo, entre outras. Os comandos abaixo podem ser utilizados para realizar movimentos simples do cursor:

- **h**: um espaço à esquerda.
- **j**: uma linha abaixo.
- **k**: uma linha acima.
- **l**: um espaço à direita.

É possível, também, utilizar as teclas de setas para movimentar o cursor. Apesar de mostrar melhor a direção de movimento, as setas não são suportadas por todos os terminais. Com o tempo e a experiência, o usuário do Vi verá que é muito prático movimentar o cursor utilizando as teclas *h*, *j*, *k* e *l*, mantendo os dedos nas posições corretas do teclado. Utilizando as teclas *h* ou *l* pode-se movimentar o cursor do início ao final da linha. Ao chegarmos ao início ou ao fim de uma linha o cursor para de se mover, não permitindo a mudança de linha e, se tentarmos avançar o cursor, um bip é emitido, indicando que a ação não pode ser executada. Para mover o cursor uma linha abaixo ou acima, é necessário utilizar as teclas *j* ou *k*, que também estão limitadas a movimentar o cursor dentro das linhas do texto, não permitindo mover o cursor acima da primeira linha ou abaixo da última linha do arquivo. Podemos utilizar números combinados com os comandos de movimentação, multiplicando a sua ação, como no exemplo da Figura 5.2, onde o comando *4l* move o cursor quatro posições para a direita, executando a mesma ação que utilizar a tecla *l* quatro vezes.

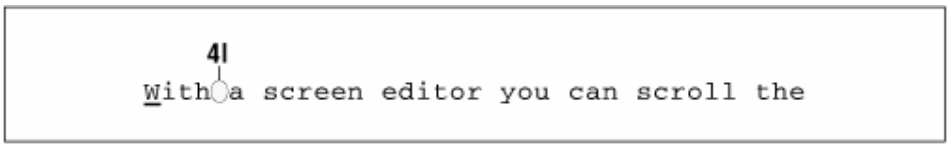


Figura 5.2
Movimentando o cursor.

A Figura 5.3 mostra os movimentos do cursor, de sua posição original (caractere "s" na décima sétima coluna da terceira linha) para as posições marcadas pelos círculos, utilizando diversos comandos do Vi.

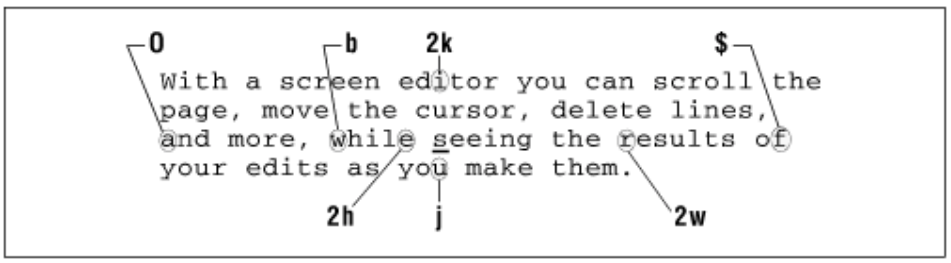


Figura 5.3
Comandos de movimentação do cursor.

Alguns comandos de movimentação do cursor:

- **0**: move o cursor para o início da linha.
- **$**: move o cursor para o final da linha.
- **w**: move o cursor de palavra em palavra para a direita, contando símbolos e pontuações equivalentes a palavras.
- **W**: move o cursor de palavra em palavra para a direita, sem contar símbolos e pontuações.
- **b**: move o cursor de palavra em palavra para a esquerda, contando símbolos e pontuações equivalentes a palavras.
- **B**: move o cursor de palavra em palavra para a esquerda, sem contar símbolos e pontuações.

Para mover o cursor por blocos de texto como palavras, frases ou parágrafos, utilizam-se os comandos:

- **0**: move o cursor para o início da linha.
- **$**: move o cursor para o final da linha.
- **w**: move o cursor uma palavra para a direita.
- **W**: move o cursor uma palavra para a direita sem contar símbolos e pontuações.
- **b**: move o cursor uma palavra para a esquerda.
- **B**: move o cursor uma palavra para a esquerda sem contar símbolos e pontuações.

Quando salvamos um arquivo, o Vi mostra na parte inferior esquerda da tela o nome e o número de linhas e de caracteres desse arquivo. É importante saber que uma linha não é necessariamente limitada ao tamanho visível na tela do monitor. Para finalizarmos uma linha utilizando o Vi, devemos utilizar a tecla *Enter* no modo de inserção. Se digitarmos caracteres indefinidamente em uma linha, sem teclar *Enter*, o Vi considerará que estes caracteres formam uma única linha, mesmo que ela seja mostrada na tela como se fossem várias linhas.

Editando texto

Comandos para edição de texto:

- **i**: inserir texto.
- **a**: anexar texto.

- **c**: alterar texto.
- **d**: remover texto.
- **x**: remover texto.
- **y**: copiar texto.
- **p**: colar texto copiado ou removido.

Ao editarmos um arquivo, necessitamos fazer diversas operações como inserção, remoção, substituição, entre outras. Os comandos a seguir são utilizados para realizar algumas das mais comuns alterações em um arquivo texto.

- **i**: inserir texto.
- **a**: anexar texto.
- **c**: alterar texto.
- **d**: remover texto.
- **x**: remover texto.
- **y**: copiar texto.
- **p**: colar texto copiado ou removido.

A Figura 5.4 mostra um texto com diversos erros e as correções necessárias. O Vi permite fazer essas correções no texto utilizando teclas básicas de edição:

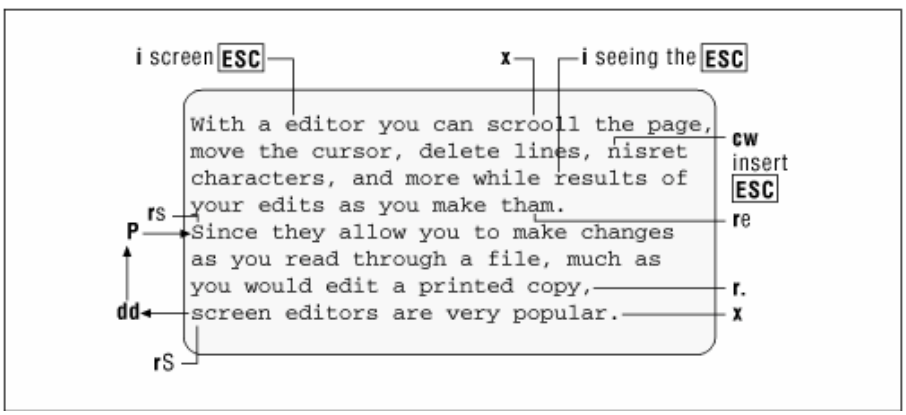


Figura 5.4
Comandos de edição.

Substituindo texto

O comando *c* permite fazer alterações utilizando combinações de comandos de movimentação do cursor como pode ser visto nos exemplos:

cw: substituir uma palavra ou parte de uma palavra onde o cursor estiver posicionado. Se o cursor estiver posicionado no meio da palavra, ela será substituída da posição do cursor até o fim da palavra. Por exemplo: para trocar "fazer" por "fazendo", devemos posicionar o cursor na letra "r", digitar o comando "cw" e inserir as letras "ndo".

cc: substituir a linha inteira onde o cursor está posicionado. Para executar esse comando, o cursor pode estar posicionado em qualquer local da linha que será substituída.

cb: substituir uma palavra ou parte de uma palavra onde o cursor estiver posicionado. Se o cursor estiver posicionado no meio da palavra, ela será substituída da posição do cursor até o início da palavra. Por exemplo: para substituir a palavra "fazendo" pela palavra "trazendo", devemos posicionar o cursor na letra "z", digitar o comando *cb* e inserir as letras "tra".

c$: altera o texto da posição do cursor até o fim da linha.

c0: altera o texto da posição do cursor até o início da linha.

O comando *c* permite fazer alterações no texto mediante combinações de comandos de movimentação do cursor, como:

- **cw**: altera o texto da posição do cursor até o fim da palavra.
- **cb**: altera o texto da posição do cursor até o início da palavra.
- **cc**: permite substituir a linha inteira onde o cursor está posicionado.
- **c$**: altera o texto da posição do cursor até o fim da linha.
- **c0**: altera o texto da posição do cursor até o início da linha.

Após digitar o comando de alteração, podemos inserir qualquer texto sem limite de caracteres. A inserção ou modificação de texto também pode ser feita com os seguintes comandos:

r: substituir o caractere onde o cursor está posicionado, sem sair do modo de comando. Esse comando pode ter seu efeito multiplicado se for precedido por algum número. Por exemplo, para substituir as próximas três letras a contar da posição do cursor, o comando *3r* deve ser utilizado. É muito útil para corrigir erros de digitação no texto, evitando o uso de *cw*, que obriga a digitação de parte da palavra novamente.

s: substituir o caractere onde o cursor está posicionado, entrando automaticamente no modo de inserção. Esse comando pode ter seu efeito multiplicado se for precedido por algum número. Por exemplo, para substituir as próximas três letras a contar da posição do cursor, o comando "3s" deve ser utilizado. A diferença desse comando para o anterior é que ele entra no modo de inserção, enquanto o outro permanece no modo comando.

C: substituir o texto de uma linha a partir da posição do cursor até o final da linha, funcionando da mesma forma que o comando composto "c$".

R: substituir texto, de forma que cada caractere digitado se sobrepõe ao caractere original. Esse modo de substituição altera no máximo uma linha, substituindo caractere por caractere.

S: alterar linhas inteiras, removendo a linha independentemente de onde o cursor estiver posicionado. O número que preceder esse comando determina quantas linhas serão substituídas. Para retornar ao modo comando, deve-se teclar "Esc" após terminar a substituição.

~: alterar o caractere onde o cursor está posicionado de minúscula para maiúscula e vice-versa. É possível, também, utilizar números multiplicadores precedentes ao comando de modo que altere vários caracteres.

A inserção ou modificação de texto também pode ser feita com os seguintes comandos:

- **r**: substituir caracteres sem entrar no modo de inserção.
- **s**: substituir caracteres entrando automaticamente no modo de inserção.
- **C:** substituir o texto de uma linha a partir da posição do cursor até o final da linha.
- **R**: substituir o texto, de forma que cada caractere digitado se sobrepõe ao caractere original. Só pode substituir caracteres da linha corrente.
- **S**: alterar linhas inteiras, removendo a linha onde o cursor estiver posicionado.
- **~**: alterar o caractere onde o cursor está posicionado de minúscula para maiúscula e vice-versa.

Exercício de fixação 1
Primeiro arquivo com Vi

Acesse o terminal e crie no diretório home do seu usuário o primeiro arquivo chamado "Arquivo-1.txt". Escreva o texto abaixo tentando utilizar os atalhos citados.

```
Qual o seu nome?

- Meu nome é Aluno.
```

Acesse o manual do editor Vi através do comando *# man vi.*

Removendo texto

- **dl**: remove o caractere onde o cursor está posicionado.
- **dd**: remove a linha corrente.
- **dw**: remove uma palavra da posição do cursor em diante.
- **db**: remove uma palavra da posição do cursor para trás.
- **d0**: remove os caracteres de uma linha da posição do cursor até o início da linha.
- **d$**: remove os caracteres de uma linha da posição do cursor até o final da linha.

Com o comando *d* podemos remover qualquer texto no arquivo editado. Assim como o comando *c*, o comando *d*, de remoção, necessita ser acompanhado de uma tecla que definirá a abrangência do texto que será removido. A seguir são vistas as opções que podem ser utilizadas com o comando *d*.

dl: remove o caractere onde o cursor está posicionado. É possível, também, utilizar números multiplicadores precedentes ao comando de modo que ele remova diversas linhas.

dd: remove a linha corrente. É possível, também, utilizar números multiplicadores precedentes ao comando de modo que ele remova diversas linhas.

dw: remove uma palavra. O comando *dw*, executado no meio de uma palavra, remove apenas a parte final dessa palavra. É importante notar que o espaço em branco após a palavra também é removido.

de: remove os caracteres de uma palavra, mantendo o(s) espaço(s) em branco após a palavra.

dE: remove os caracteres até o final da palavra incluindo a pontuação, se houver.

db: remove os caracteres de uma palavra da posição do cursor até o início da palavra.

d0: remove os caracteres de uma linha da posição do cursor até o início da linha.

d$: remove os caracteres de uma linha da posição do cursor até o final da linha.

Rolagem do arquivo

- **Ctrl+F**: rola a tela inteira para a frente no texto.
- **Ctrl+B**: rola a tela inteira para trás no texto.
- **Ctrl+D**: rola a metade da tela para a frente no texto.
- **Ctrl+U**: rola a metade da tela para trás no texto.
- **Ctrl+E**: rola a tela uma linha para cima.
- **Ctrl+Y**: rola a tela uma linha para baixo.
- **zEnter:** rola a linha corrente para o topo da tela.

- **gg**: move o cursor para a primeira linha do arquivo.
- **G**: move o cursor para a última linha do arquivo.

Durante a inserção de texto, quando chegamos ao final da última linha da parte inferior da tela, a primeira linha da parte superior deixa de ser mostrada, todas as linhas sobem na tela e uma nova linha passa a ser mostrada na posição da linha inferior. Essa movimentação é chamada de scrolling ou rolagem de tela. Alguns comandos para rolamento de tela:

Ctrl+F: rola a tela inteira para a frente no texto.

Ctrl+B: rola a tela inteira para trás no texto.

Ctrl+D: rola a metade da tela para a frente no texto.

Ctrl+U: rola a metade da tela para trás no texto.

Ctrl+E: rola a tela uma linha para cima.

Ctrl+Y: rola a tela uma linha para baixo.

zEnter: rola a linha corrente para o topo da tela.

gg: move o cursor para a primeira linha do arquivo.

G: move o cursor para a última linha do arquivo.

O comando *z* pode ser precedido pelo número da linha que será utilizada no lugar da linha corrente. Por exemplo, o comando *200zEnter* rola a linha 200 do texto para o topo da tela, e não a linha corrente. Existem outros comandos de rolagem além dos comandos acima, que podem ser consultados em guias de referência do Vi na internet.

Se abrirmos o Vi no ambiente gráfico, utilizando o Xterm, a exibição do texto do arquivo será feita dentro da janela do Xterm, que pode ou não estar ocupando toda a tela do monitor. Quando o número de linhas do texto for maior que o número de linhas configurado na janela, aparecerão no lado direito da janela botões de scrolling. Esses botões permitem o rolamento do texto, utilizando-se um mouse.

Comando de busca

- O comando de busca é o caractere */*, que mostra uma barra na linha inferior da tela, onde deve ser digitado o texto que será o padrão a ser procurado.
- Este padrão de busca pode ser uma palavra ou qualquer sequência de caracteres. Os espaços em branco também são considerados parte do padrão.
- O Vi inicia a procura do padrão na posição do cursor, seguindo adiante no arquivo, e ao chegar ao final volta ao início do arquivo e continua a procura até a posição do cursor.
- O cursor é movido para a posição da primeira ocorrência do padrão procurado. Se não há nenhuma ocorrência do padrão, a mensagem “Pattern not found” é mostrada na parte inferior esquerda da tela.

O comando de busca é o caractere */* que, quando teclado, mostra uma barra na linha inferior da tela onde deve ser digitado o texto que se quer procurar. Esse padrão de busca pode ser uma palavra ou qualquer sequência de caracteres. Os espaços em branco também são considerados como parte do padrão. O Vi inicia a procura do padrão na posição do cursor, seguindo adiante no arquivo e, ao chegar ao final, volta ao início do arquivo e continua a procura até a posição do cursor. Se o padrão for encontrado, o cursor é movido para a posição de sua primeira ocorrência.

Se não há nenhuma ocorrência do padrão, a mensagem “Pattern not found“ é mostrada na parte inferior esquerda da tela. Na verdade, essa mensagem pode variar dependendo da versão do Vi ou do Linux, mas o significado será o mesmo.

Para realizar a procura no sentido inverso, voltando sobre o texto, devemos utilizar o comando *?*. Nesse caso, a pesquisa é feita até o início do texto e reiniciada no final, até chegar à posição do cursor. Os comandos a seguir podem ser utilizados para repetição da última busca com o último padrão utilizado. Isso evita a digitação de todo o padrão novamente, o que é útil para encontrar diversas ocorrências repetidas de um mesmo padrão.

n: repete a busca na mesma direção.

N: repete a busca na direção oposta.

/Enter: repete a busca adiante no texto.

?Enter: repete a busca voltando no texto.

Os comandos de repetição de busca permitem achar padrões, realizar alterações e iniciar novas buscas, facilitando substituições em todo o texto. Pode-se combinar simultaneamente o comando de busca com o de substituição para alterar todas as ocorrências de uma palavra por outra. O comando abaixo faz uma busca partindo da primeira linha (1) até a última ($) e substitui as ocorrências da palavra “umapalavra” pela palavra “novapalavra”.

```
:1,$s/umapalavra/novapalavra
```

- Para realizar a procura no sentido contrário, voltando sobre o texto, devemos utilizar o comando *?*.
- Comandos para repetição em buscas:
  - **n:** repete a busca na mesma direção.
  - **N**: repete a busca na direção oposta.
  - **/Enter**: repete a busca adiante no texto.
  - **?Enter**: repete a busca voltando no texto.
- Combinando o comando de busca com o de substituição podemos, por exemplo, alterar todas as ocorrências de uma palavra por outra em todo o texto como no exemplo: `:1,$s/umapalavra/novapalavra`

Exercício de fixação 2
Busca no arquivo com Vi

Acesse o arquivo criado no exercício anterior e teste os comandos de busca utilizando todos os recursos citados.

Comandos combinados

Conhecemos, nos itens anteriores, algumas combinações de comandos, utilizando também números multiplicadores. Na Tabela 5.1 são mostrados exemplos de combinações possíveis de comandos e suas ações correlatas.

Alteração	Remoção	Cópia	Do cursor a ...
cH	dH	yH	primeira linha da tela
cL	dL	yL	última linha da tela
c+	d+	y+	próxima linha
c5l	d5l	y5l	coluna 5 da linha corrente
c/padrão	d/padrão	y/padrão	padrão
cn	dn	yn	próximo padrão
cG	dG	yG	última linha do arquivo
c13G	d13G	y13G	linha 13 do arquivo

Tabela 5.1
Comandos combinados.

Vimos que o Vi é um editor simples, porém muito poderoso, que nos oferece grande flexibilidade para a edição de textos. Entendemos, agora, porque é um dos editores preferidos dos administradores de sistemas Unix-like.

Roteiro de Atividades 5

Atividade 5.1 – Criando um texto no Vi

Crie o arquivo "atividades" utilizando o Vi contendo o seguinte texto:

Brasília tem caso suspeito de parasitose do peixe cru

BRASÍLIA – A Secretaria de Saúde do Distrito Federal detectou o primeiro caso suspeito de difilobotríase, parasitose intestinal transmitida pela ingestão de peixe cru ou mal cozido, que já atingiu pelo menos 27 pessoas em São Paulo nos últimos doze meses. A possível vítima, uma mulher de 25 anos, teria se contaminado em outro estado. A vigilância sanitária local prometeu reforçar os cuidados nas operações de fiscalização a estabelecimentos comerciais, pretendendo centrar as atenções na forma de armazenamento dos peixes.

A suspeita de contaminação foi notificada pela Secretaria de Saúde local no último dia 6. O laudo do laboratório contratado pela Secretaria está sendo aguardado para os próximos dias, com a confirmação da presença do parasita no organismo da vítima. O infectologista Alexandre Cunha, que atendeu a paciente, afirmou que fez uma análise morfológica do parasita encontrado em amostra das fezes da vítima e constatou tratar-se do causador da difilobotríase.

"Ainda não temos certeza da contaminação. Esse tipo de parasita não é comum no Brasil. Mas o congelamento do peixe de forma adequada impede a transmissão", explicou a diretora de vigilância epidemiológica local, Disney Antezana.

A Agência Nacional de Vigilância Sanitária (Anvisa) recomenda que se evite o consumo de peixes crus ou se certifiquem de que o produto esteve congelado em pelo menos 20 graus negativos por, no mínimo, sete dias. Para matar o transmissor, também é possível manter o peixe em 35 graus negativos por 15 horas.

Carolina Brígido, O Globo

> Os caracteres acentuados podem ser digitados sem acento, caso o sistema operacional não possua suporte à acentuação.

Atividade 5.2 – Usando recursos básicos do Vi

Utilize os comandos básicos do Vi, para realizar funções simples de cada vez em sequência, ou seja, movimentação, inserção, remoção ou busca, para executar as seguintes edições no texto:

a. Pesquise o número de ocorrências da palavra "que".
b. Apague as linhas da posição do cursor até o final do arquivo.
c. Movimente a linha 2 para o final do texto.
d. Coloque o nome "Anvisa" em maiúsculas.
e. Apague os 15 primeiros caracteres da linha 3. Como essa ação poderia ser feita utilizando outro comando?
f. Volte ao prompt do Linux sem salvar as modificações no texto.

Atividade 5.3 – Combinando recursos do Vi

a. Encontre e altere as ocorrências da palavra “que”, colocando-as em maiúsculas, executando uma combinação de comandos que faça esta ação de uma só vez.

b. Remova do quinto caractere da linha 1 até o final desta linha.

c. Substitua todos os espaços em branco do texto pelo caractere “_”.

d. Sem sair do Vi, informe quantos bytes tem o arquivo que está sendo editado.

e. Apague as cinco primeiras palavras da linha 1.

f. Volte ao prompt do Linux sem salvar as modificações no texto.

Atividade 5.4 – Execução de comandos diversos

a. Executando um comando de movimentação, vá para a linha 3 e apague as linhas 3 e 4.

b. Mova as linhas da posição atual do cursor até a última linha para o início do texto.

c. Desfaça as duas últimas alterações.

d. Refaça somente a última alteração.

e. Copie as duas primeiras linhas e cole-as no fim do arquivo.

f. Volte ao prompt do Linux sem salvar as modificações no texto.

Atividade 5.5 – Execução de comandos avançados

a. Abra o arquivo “atividades”, de forma que o cursor fique posicionado na linha 3.

b. Utilize um único comando para apagar todas as linhas do texto que não possuam a palavra Anvisa.

c. Salve o arquivo com o mesmo nome no diretório */tmp*.

d. Remova todas as linhas que contenham a palavra “peixe”, utilizando um único comando.

e. Sem sair do arquivo “atividades”, abra na mesma janela do Vi o arquivo */etc/profile*.

f. Volte ao prompt do Linux salvando as modificações feitas no arquivo “atividades”.

6

Shell

objetivos

Conhecer o funcionamento do Shell e aprender a lidar com seus processos.

conceitos

Shell, gerenciamento, criação e execução de processos, ambientes e suas variáveis e noções de Shell Script.

Noções básicas

Shell é um ambiente de interpretação de comandos. Ele é a ponte entre o usuário e o sistema operacional, ou seja, é através dele que o usuário requisita ações ao sistema, utilizando-se de comandos. Podemos observar a atuação do Shell quando abrimos um terminal ou console e executamos comandos como *ls*, *cat*, *touch*, *mkdir*, *cp*, *rm*, *mv* etc.

A interação do usuário com o Shell adquire a forma de um diálogo. Primeiramente, o Shell solicita uma entrada ao usuário, mostrando um prompt na tela do terminal. Assim que um comando é digitado, o Shell o executa. Este comando pode fazer parte de seus comandos internos (built in) ou ser algum outro comando/programa do sistema operacional. Quando a tarefa do comando foi completada, o Shell, uma vez mais, solicita nova entrada ao usuário, mostrando novamente o cursor, de modo que se possa continuar digitando comandos e interagindo com o Shell. A Figura 6.1 mostra a sequência de passos em um diálogo do usuário com o Shell:

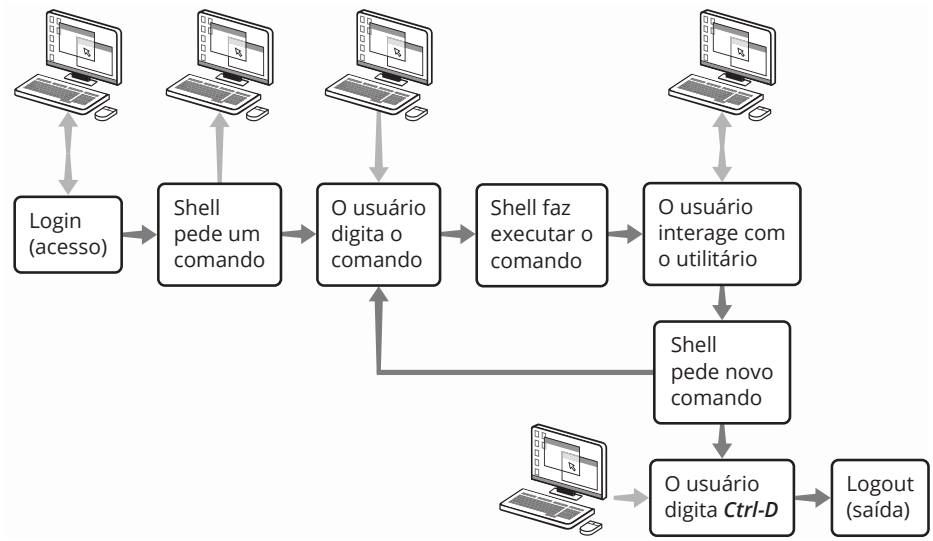


Figura 6.1
Diálogo do usuário com o Shell.

O Shell é tão poderoso que suporta várias funcionalidades como:

- Redirecionamento de entrada e saída de aplicações, como por exemplo:

```
# ls /home > saida
```

- Execução de sequências de comandos:

```
# cat /etc/passwd | grep root
```

- Escrita na saída padrão ou redirecionada.

```
# echo "Este é um curso de Introdução ao Linux".
```

- Automatização de sequências de comandos (programação).

Ele possui ainda uma poderosa ferramenta de programação, permitindo a automatização de tarefas complexas por usuários ou administradores. A não ser que seja feito um redirecionamento, a entrada padrão do Shell é o teclado e a saída é o monitor. Podemos fazer com que o Shell escreva qualquer coisa na saída padrão, ou saída redirecionada, através do comando *echo*. Essa informação pode ser, por exemplo, um texto:

```
# echo "Este é um curso de Introdução ao Linux"
```

O uso das aspas é opcional. No entanto, convenciona-se usá-las para indicar ao Shell que não é necessário interpretar aquele conteúdo, basta mostrá-lo na saída desejada. Essa convenção será obrigatória em outras ocasiões.

Cada conta de usuário, quando criada, recebe um Shell padrão, através do qual aquele usuário fará a interação, em modo texto, com o Sistema Operacional. Para o Linux, são vários os Shells disponíveis. Os mais comuns são os da família *Bourne Shell*, dentre eles o *Bourne Shell* (sh) e o *Bourne Again Shell* (bash). O *bash* é o mais usado nas distribuições Linux. Há ainda os da família *C Shell*, como o *csh*, que possui suporte à computação numérica, bastante deficitário na família Bourne, mas com a desvantagem de ter uma sintaxe parecida com a linguagem C, não muito fácil de ser utilizada. A diferença entre os diversos Shells existentes está basicamente nas funcionalidades incorporadas e na sintaxe dos comandos, que podem ser simples ou mais complexas.

Para saber qual Shell está configurado para o seu usuário, basta verificar no arquivo */etc/passwd*. E para saber os Shells que estão disponíveis no seu Linux, basta verificar o arquivo */etc/shells*. Este curso é focado no Shell bash.

Gerenciamento de processos

O que é um processo?

- Um programa em execução.

Duração de um processo:

- Enquanto houver instruções para serem executadas.

Recursos e gerenciamento:

- Os processos utilizam recursos como processador e memória, e são organizados e gerenciados pelo Sistema Operacional.

Processo INIT:

- Processo pai de todos os outros processos.

Escalonamento:

- Baseado em time-sharing.

Um processo é um programa em execução. A "vida" de um processo começa no início de um programa e dura enquanto as instruções do programa continuarem a especificar novas operações. Quando o programa chega ao final, isto é, o sistema operacional termina a execução de suas instruções, o processo relativo a este programa "morre".

O processo utiliza recursos do computador, como processador e memória, para realizar suas tarefas. É de responsabilidade do sistema operacional organizar e gerenciar todos os processos. Quando o sistema é iniciado, a função "start kernel" cria o processo número zero, que é uma *thread* que gerará todos os demais processos. Esta função chama a função "init", que será o processo número um. O processo INIT será o pai de todos os processos e um dos últimos a morrer.

O conceito de *thread* está associado às máquinas multiprocessadores e atividades concorrentes. Usualmente é definido como um fluxo de controle no interior de um processo. Quando dois processos compartilham as mesmas estruturas, eles atuam como se fossem diferentes *threads* no interior de um único processo. No Linux, *threads* e processos são tratados da mesma forma.

O gerenciamento de processos permite a estruturação dos programas executados pelo sistema. Quem decide que processo deve ser executado a cada momento é o escalonador. Ele faz isso de forma que não seja desperdiçado tempo de hardware, garantindo a eficiência do sistema. O escalonador do Linux é baseado em *time-sharing*, ou seja, o tempo do processador é dividido em fatias de tempo (quantum) em que são alocados os processos. Se durante a execução de um processo esgota-se o quantum, um novo processo é selecionado para execução, através das regras de escalonamento, que se baseiam nas prioridades atribuídas a cada processo. Como o quantum é bastante pequeno, tem-se a impressão de que o sistema operacional executa vários processos ao mesmo tempo, característica chamada de multitarefa.

Requisições feitas pelos processos ao Sistema Operacional:

- Criar processos, gerenciar memória, ler e escrever arquivos etc.
- Exemplos: fork, exec, exit, kill.
- As chamadas de sistema alteram o ciclo normal de vida de um processo.

Como é feito o controle de um processo?

- Através de características como estado, prioridade de execução, recursos de memória etc.
- O processo é identificado pelo PID e PPID (processo-pai que o criou).
- # ps -f (visualiza essas informações)

Existem diversas chamadas de sistema no Linux que alteram o ciclo normal de "vida" de um processo. As chamadas de sistema são requisições feitas pelos processos ao sistema operacional por recursos aos quais ele não consegue acessar. As chamadas são usualmente para criar processos, gerenciar memória, ler e escrever arquivos e fazer entrada e saída. Estão entre as mais comuns: *fork*, *exec*, *exit*, *kill*.

O controle dos processos é feito através de um conjunto de características como proprietário do processo, seu estado (se está em espera, em execução etc.), prioridade de execução e recursos de memória. Para que se viabilize o controle, cada processo é identificado com um único número chamado de "process identifier" ou PID. Cada processo possui um processo-pai (exceto o processo *init*), que é o PID do processo que o criou, chamado de "parent process identifier" ou PPID. O comando *# ps -f* permite visualizar esses números.

Permissões – controladas pelo UID e GID:

- Root – UID = 0
- Para usuário ter privilégios de root – GID = 0
- Visualizados no arquivo */etc/passwd*.
  - Ex: root:x:0:0:root:/root:/bin/bash - os números são UID e GID, respectivamente.

O controle das permissões dos processos funciona usando a mesma lógica. Os administradores do sistema atribuem números para os usuários ("user identifier" ou UID) e para grupos ("group identifier" ou GID), quando são criadas as contas de usuário. O sistema usa o UID e o GID de um usuário para recuperar informações de sua base de dados, relativas aos privilégios permitidos para o usuário. O usuário de maior privilégio é o root, também conhecido como *superusuário*, que tem o UID igual a 0 (zero) em uma escala que vai além de 65 mil, dependendo do sistema operacional. O usuário root é usualmente o administrador do sistema, possuindo plenos poderes. Cada processo pertence a um usuário e também pertence ao seu grupo. Para fazer com que um usuário tenha os mesmos privilégios que o root, é necessário que o GID dele seja 0. No arquivo */etc/passwd* é possível visualizar o UID (primeiro número que aparece) e o GID (segundo número) de cada usuário criado; se não for especificado, o sistema operacional cria um grupo para cada usuário.

Criação de processos

Os processos são criados a partir de outros processos (processos-pai).

- Processo inicial – INIT.

A criação se dá através de:

- Duplicação de um processo já existente.
  - Operação *fork-exec*.
  - Execução concorrente.
  - fork – duplica o processo.
  - exec – substitui o código antigo pelo novo no processo criado.

No Linux, somente o processo INIT é criado a partir do zero. Todos os demais processos são criados a partir dele. A criação de um processo é baseada em uma operação do tipo *fork-exec*, na qual um processo já existente se duplica através de uma chamada de sistema chamada *fork,* e em seguida substitui seu código por outro, através da chamada de sistema *exec*, permitindo que o código do filho seja executado sem que o código dele seja prejudicado.

Isso acontece quando digitamos um comando no Shell. O bash, no nosso caso se duplica usando *fork* e então chama o *exec* para sobrepor sua memória com os conteúdos do comando solicitado. Essa combinação permite que os processos gerados sejam concorrentes, ou seja, executem de forma independente uns dos outros. A Figura 6.2 ilustra outro exemplo típico, no qual um programa de menu oferece uma opção de três telas de entrada de dados. Quando o usuário faz uma seleção, o programa de menu bifurca um processo-pai que executa o programa escolhido, e o programa de entrada de dados roda até a sua conclusão. Enquanto isso, o programa de menu pode ser usado para outra opção, o que gera uma nova bifurcação.

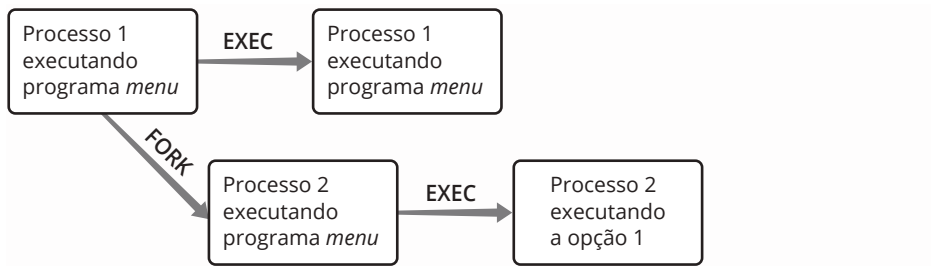


Figura 6.2
Execução concorrente.

No *fork,* o PID de cada um dos processos clonados é diferente. No entanto o PPID é igual (observe a árvore hierárquica dos processos e seus PIDs via comando *# pstree –p*). No *exec*, como ele faz simplesmente uma sobreposição de conteúdo, os PIDs dos dois processos são iguais. Existe ainda a função *clone*, semelhante à *fork*, mas com algumas características diferentes.

Da mesma forma, é possível executar processos de forma sequencial, sem que haja uma bifurcação ou duplicação de código. Através da chamada de sistema *exec,* o kernel carrega o novo programa na memória e inicia a execução do novo processo especificado pelo *exec.*

Este novo programa começa a execução no mesmo ambiente do seu predecessor.
Os arquivos abertos pelo programa anterior permanecem abertos para o novo processo, que também retém as identidades do usuário e do grupo de usuários. A Figura 6.3 mostra um exemplo simples de como a sequência de programas de impressão de relatório pode ser executada usando-se *exec.*

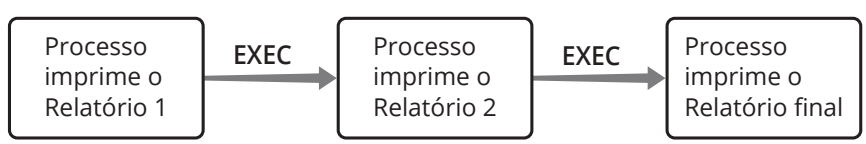


Figura 6.3
Execução sequencial.

O primeiro processo imprime o relatório antes de *exec* executar o segundo programa de impressão de relatório. O segundo programa acrescenta seus relatórios, chama o terceiro programa para imprimir o relatório final e, finalmente, fecha os arquivos e termina o processo. As chamadas *exec* são usadas muito similarmente ao encadeamento de programas.

A função *exec* faz com que o processo por ele chamado substitua o processo em execução. Assim, qualquer sequência que houver neste processo após o comando *exec* não será executada.

Processos em background e daemons

Daemon é um processo executado em background.

- Geralmente não tem tempo de vida definido.

Tarefas nas quais os daemons são usados:

- Paginação de memória.
- Solicitações de login.
- Manipulação de e-mails.
- Transferência de arquivos.
- Coletas de estatísticas de operações do sistema.
- Solicitações de impressão.

Quando um comando é digitado pelo usuário por intermédio da entrada padrão (o teclado), o Shell solicita ao kernel do sistema operacional a execução deste comando, aguarda pela sua finalização e exibe o resultado do comando na tela, voltando a aguardar a entrada de novos comandos. Quando a entrada e saída padrão são o teclado e o terminal de vídeo, o usuário pode terminar a execução de um comando digitando o caractere de interrupção, por default *Ctrl-C* ou ^*C*. Este tipo de execução de processo pelo Shell é chamado de *foreground* ou "em primeiro plano". Pode-se também executar o mesmo processo em *background* ou "em segundo plano", de forma que o usuário não precisa ficar aguardando pela finalização do processo para entrar com um novo comando, pois o prompt é automaticamente liberado para a entrada de novos comandos. Utiliza-se o caractere & para a execução de um comando em *background*. A sintaxe utilizada é a seguinte:

```
# comando &
```

A execução de um comando em background permite ao usuário executar outros comandos simultaneamente. Estes comandos são executados imediata e independentemente de já haver um processo em execução. Outra vantagem é utilizar a execução em *background* para realizar tarefas demoradas, como a impressão de arquivos extensos ou a execução de *sort* ou *find* em arquivos grandes, ou ainda programas que serão executados por algum tempo como, por exemplo, um navegador web, um editor de textos etc.

Entretanto, deve-se tomar cuidado ao rodar scripts em background, pois a saída continua a ser a padrão, e podem aparecer resultados do programa na tela, atrapalhando a visualização de outras tarefas do usuário. É recomendável que a saída do processo em background seja redirecionada para um arquivo, usando a seguinte sintaxe:

```
#  comando >arquivo 2>&1 &

[1] pid
```

Desta forma, toda a saída-padrão e a saída de erros serão redirecionadas para o arquivo. O número do processo correspondente é mostrado logo a seguir. Este número pode ser usado para finalizar o processo ou monitorá-lo. Quando o processo termina, uma mensagem é mostrada assim que pressionamos uma tecla qualquer, geralmente a tecla *Enter*. A mensagem tem a seguinte forma:

```
[1]+ Done comando
```

- Cron: usado para programar tarefas.
- Editando o crontab do usuário, é possível programar tarefas.

```
# crontab -e
```

- Na última linha do arquivo programa-se o comando. Exemplo:

```
03 17 * * * cp /etc/passwd /home/USER/passwd
```

- Onde 03 são os minutos e 17, a hora.

Um daemon é um processo executado em background e normalmente não tem tempo de vida definido, podendo ser executado por tempo indeterminado. O sistema operacional Linux utiliza muitos daemons para realizar rotinas e tarefas, como a paginação da memória, as solicitações de login, a manipulação de e-mails, a transferência de arquivos, a coleta de estatísticas de operação do sistema e as solicitações de impressão.

Um daemon interessante é o cron. Ele acorda uma vez por minuto para ver se existe algum trabalho a ser feito. Caso exista, ele o faz. Depois volta a dormir até o momento da próxima verificação.

Para programar as tarefas que devem ser realizadas pelo cron, basta editar o crontab do usuário através do comando:

```
# crontab -e
```

E seguir a sintaxe especificada no próprio arquivo, agendando minutos, horas, dias, mês, dia da semana e tarefa a ser executada (lembre-se de deixar uma linha em branco no final do arquivo, para garantir que o último comando seja lido). Assim, é possível automatizar qualquer tarefa, como um backup, por exemplo.

Para ver as tarefas agendadas utilize o comando *# crontab –l*. Se o agendamento não funcionar, reinicie o cron com o comando *# sudo restart cron.*

Sinais do sistema

Um sinal é uma notificação de software enviada por um processo ou pelo sistema operacional relativo a um evento ocorrido. Sinais são usados para passar informações.

- ABRT: aborta o processo.
- INT: sinal de “Ctrl+C”.
- Outro processo pode usar o sinal para realizar alguma ação, se ele ativar seu manipulador de sinais.

Alguns sinais são gerados por comandos no Shell. Exemplo:

```
# kill -9 pid
```

- Usado para terminar a execução de um processo.

Para manter a execução de um processo após o logout:

```
# nohup arquivo
```

- A saída do arquivo é redirecionada para o arquivo *nohup.out* no diretório *home* do usuário.

Os processos concorrentes são de natureza assíncrona, ou seja, não há uma ligação necessária entre a execução destes processos. Alguns processos podem necessitar de informações sobre o estado de outros processos que estão sendo executados simultaneamente. Para passar informações entre processos, são utilizados sinais; um sinal é uma notificação de software enviada por um determinado processo ou pelo sistema operacional, relativo a um evento neles ocorrido. Outro processo pode utilizar este sinal para realizar uma ação. O tempo de vida de um sinal é o intervalo entre sua geração e seu envio. Um sinal que foi gerado, mas não foi enviado, é um sinal pendente. Um determinado processo recebe um sinal se tiver ativado um manipulador de sinais, quando o sinal for enviado. Um programa ativa um manipulador de sinais através de uma chamada de sistema com o nome do programa do usuário. Um processo pode temporariamente bloquear o recebimento de um sinal, de forma que este sinal bloqueado não afete a execução do processo, mesmo que seja enviado. Bloquear um sinal é diferente de ignorar um sinal. Quando um processo bloqueia um sinal, o sistema operacional não envia o sinal até que o processo o desbloqueie. De outra forma, um processo pode ignorar um sinal em vez de bloqueá-lo; neste caso, o processo manipula o sinal recebido e o descarta, sem utilizá-lo. Na Tabela 6.1 são apresentados alguns desses sinais.

Alguns sinais também são gerados por comandos no Shell, como o comando *kill*, utilizado quando queremos terminar um processo. Ele utiliza o número associado ao processo para terminá-lo. A sintaxe do comando é:

```
# kill pid
```

Após a digitação, ao teclar "Enter", a seguinte mensagem será mostrada informando que o processo foi terminado:

```
[1]+ Terminated process
```

Se nenhuma mensagem de indicação de finalização do processo for mostrada, aguarde um minuto, pois a terminação do processo pode estar sendo realizada. Caso haja demora excessiva, pode-se executar outra forma do comando *kill*:

```
# kill -9 pid

[1]+ Killed process
```

Este comando fará com que a terminação do processo seja realizada de maneira imediata, enquanto outros processos em background continuam a ser executados após o logout.

Sinal	Exemplo
ABRT	Aborta o processo.
ALRM	Sinal do temporizador da chamada de sistema *alarm*.
CHLD	Termina, interrompe e continua o "processo-filho".
CONT	Continua a execução do processo interrompido.

Sinal	Exemplo
FPE	Sinaliza erro em operação aritmética, como a divisão por zero.
ILL	Indica instrução inválida.
INT	Sinal de Ctrl-C.
KILL	Indicação de processo terminado, que não pode ser ignorada.
PIPE	Saída de um pipe não recebida.
SEGV	Referência de memória inválida.
STOP	Sinal de interrupção de processo de execução.
TERM	Sinal de término de processo.
TSTP	Interrompe a digitação de caracteres no terminal.
TTIN	Processo em background aguardando entrada do terminal.
TTOU	Processo em background aguardando saída do terminal.
URG	Dados de alta prioridade disponíveis em um socket.

Tabela 6.1
Sinais do sistema.

Para manter a execução de um processo após o logout, deve ser usado o comando *nohup:*

```
# nohup arquivo
```

Este comando executa o arquivo de forma que não seja encerrado com a saída da sessão de trabalho. A saída do arquivo é redirecionada para o arquivo *nohup.out* no diretório *home* do usuário, o que é muito útil, por exemplo, quando se quer executar programas longos, e há necessidade de se ausentar e terminar a sessão.

Visualização de processos

Quando um processo está sendo executado, podemos usar o seu número de identificação para verificar o estado da sua execução por meio do comando *ps*.

```
# ps
# ps aux
# ps l
```

- Lista de informações: UID, PID, PPID, PRI, STAT, TIME, COMM, CPU, TTY, INTPRI, SZ e WCHAN.
- Usado também em canalizações para produção de filtros.

```
# ps aux | grep mozilla
```

Quando um processo está sendo executado, podemos usar o seu número de identificação para verificar o estado da sua execução por meio do comando *ps*. O comando *ps* sozinho, sem argumentos, mostra apenas os processos associados ao usuário. Já o comando *ps aux* mostra todos os processos em execução no sistema no momento, inclusive os *daemons*. Outra opção é o comando *ps l*, que retorna uma longa lista associada a cada processo, incluindo as seguintes informações:

- UID – identificador do usuário dono do processo.
- PID – identificador único associado a cada processo.

- PPID – identificador único associado ao processo-pai do processo em questão.
- PRI – prioridade do processo; programas de prioridade mais alta conseguem mais rapidamente a atenção da CPU.
- STAT – situação atual ou estado do processo, com as seguintes opções:
  O – não existente;
  I – processo intermediário;
  R – executando (running);
  S – dormindo (sleeping);
  W – em espera (waiting);
  Z – processo terminado;
  T – processo parado (stopped).
- TIME – tempo decorrido, desde a emissão do comando que deu origem ao processo até o momento de emissão do comando em minutos e segundos.
- COMM – nome do comando submetido pelo terminal especificado no parâmetro TTY.
- CPU – porcentagem de utilização da CPU pelo processo.
- TTY – terminal TTY ao qual foi submetido o comando que deu origem ao processo.
- INTPRI – prioridade do processo internamente ao kernel.
- SZ – tamanho em kilobytes (1.024 bytes) da imagem do processo no kernel.
- WCHAN – evento sob o qual o processo está aguardando, quando está nos estados de espera (W) ou dormindo (S).

O comando *ps* também pode ser executado em canalizações com o comando *grep*, o que facilita encontrar as informações sobre processos executados em background. Por exemplo:

```
# ps aux | grep mozilla
```

Permite mostrar apenas os dados do processo que está executando o navegador web Mozilla.

Variáveis de ambiente

São utilizadas para manipular as posições de armazenamento na memória do computador. É possível acessar essa posição, armazenando, lendo, modificando o dado lá gravado.

- Uma variável tem um nome e um valor a ela associado.
- Os conteúdos das variáveis são preenchidos através do comando de atribuição (=).

```
# teste=10              # não deve haver espaço entre os ele-
  mentos de uma atribuição
```

- Para visualizar o conteúdo de uma variável

```
# echo $teste
```

Outros exemplos:

```
# X=1                   #funciona

# X=teste              #funciona

# X=varias palavras          #erro

# X=”varias palavras”       #funciona

# X=”$Y unidades”              # o $Y (será substituído pelo valor
da variável Y)
```

```
# X='$Y unidades'        # $Y (não será substituído)
```

Quando fazemos login no sistema, várias ações automáticas são desencadeadas, tais como leituras de arquivos, execução de programas, etc. Estas ações acontecem para configurar um ambiente no qual o usuário realizará suas tarefas. A configuração padrão (*default*) do ambiente é definida previamente em alguns arquivos, por meio, entre outros, de variáveis de ambiente. Elas são usadas pelo Shell para armazenar informações de usos específicos. Na Tabela 6.2, são mostradas algumas dessas variáveis.

Tabela 6.2
Algumas variáveis de ambiente do Shell.

Variável	Descrição
PATH	Contém uma lista de caminhos a serem pesquisados em busca de comandos. Cada caminho é separado por dois pontos (:) e é definido no arquivo */etc/profile* (seja cuidadoso: a mudança neste arquivo afeta a todos os usuários).
HOME	Contém o diretório do usuário atual, local padrão destinado a armazenar suas informações. É definido no arquivo */etc/passwd*.
SHELL	Contém o tipo de Shell padrão para aquele usuário específico. Pode ser configurado no arquivo */etc/passwd*.
PWD	Contém o diretório de trabalho mais recente.
RANDOM	Gera um número inteiro entre 0 e 32.767.
IFS	Contém o separador usado para separar palavras no Shell (geralmente é espaço, tab e nova linha).

As variáveis de ambiente são sempre usadas com letras maiúsculas. O Shell é por característica sensível ao caso, ou seja, interpreta letras maiúsculas e minúsculas como sendo diferentes.

A visualização do conteúdo das variáveis de ambiente é feita através do comando *echo*. Para sabermos qual Shell está sendo usado para aquele usuário específico basta executar o comando:

```
# echo $SHELL
```

Devemos instruir o comando echo para mostrar o "conteúdo" da variável de ambiente SHELL e não a própria palavra "SHELL". Isso é indicado com o uso do caractere $ antes do nome da variável de ambiente.

O comando *set* nos permite visualizar todas as variáveis locais disponíveis no Shell corrente. Para visualizarmos as variáveis de ambiente globais do Shell corrente, devemos usar o comando *env*. O comando *set* também pode ser usado para definir o valor de uma variável de ambiente. É possível usar o comando *unset* para eliminar uma variável, em vez de defini-la com o valor "null". É uma tarefa relativamente simples customizar o seu ambiente no Linux. Você pode visualizar como está configurado o seu ambiente digitando *set*. Uma das principais variáveis de ambiente é a UID, utilizada pelo sistema para identificar os usuários e suas permissões. Algumas variáveis, no entanto, são do formato *read-only*, sendo necessárias permissões de administrador para alterá-las. Desta forma, permanecem protegidas e mantidas sem alterações após serem definidas. Para declararmos uma variável de ambiente como *read-only*, devemos usar o comando *readonly* ou o comando *declare* com a opção *-r*. O administrador pode, se necessário, tornar variáveis de ambiente *read-only* no arquivo */etc/profile*. O objetivo é fornecer uma visão do modo como o ambiente do usuário é definido por intermédio de variáveis.

Uso de aspas simples, duplas e barra invertida

São usadas para proteger caracteres, para que não sejam interpretados pelo Shell.

- A barra invertida "\" protege o caractere que vem logo após ela.

```
# touch meuarquivo\ >   ### o nome do arquivo criado será meuarquivo>
```

- As aspas simples (') protegem todos os caracteres que estão entre elas.

```
# echo 'Estamos estudando o "uso das aspas". '      # (protege as
  aspas duplas)

# echo 'Estamos estudando o 'uso das aspas'.'    # (não protege a
  si mesma)
```

As aspas duplas também são usadas para isso, mas não protegem "$" e "\".

```
# echo "Estamos estudando o 'uso das aspas'."

# echo "Estamos estudando o \"uso das aspas\"."

# echo "Conteúdo do diretório home: $HOME"

# echo "Estamos estudando o "uso das aspas"." # (não protege a si mesma)
```

As aspas simples (' ') e duplas (" ") e a barra invertida (\) são usadas quando é necessário que os caracteres por eles marcados fiquem protegidos, ou seja, não sejam interpretados pelo Shell. Assim, temos:

- A barra invertida (\) protege o caractere que vem logo após ela. Por exemplo, para criar um nome de um arquivo contendo o caractere ">" que sabemos ser de redirecionamento, temos que protegê-lo:

```
     # touch meuarquivo\ >
```

- As aspas simples (') protegem todos os caracteres que estão entre eles.

```
# echo 'Estamos estudando o "uso das apas". '      # protege as
aspas duplas

# echo 'Estamos estudando o 'uso das aspas'.'    # não protege a
sim mesma
```

- As aspas duplas também são usadas para isso, mas não protegem $, e \

```
# echo "Estamos estudando o 'uso das aspas'."

# echo "Estamos estudando o \"uso das aspas\"."

# echo "Conteúdo do diretório home: $HOME"

# echo "Estamos estudando o "uso das aspas"."      # não protege a
sim mesma
```

Exercício de fixação 1
Visualização de processos

Utilizando a página de manual do comando *ps*, descreva a diferença entre o comando *ps –aux* e *ps –ef*. Em que sintaxe cada um deles é visto?

Exercício de fixação 2
Visualização de processos em tempo real

Qual comando é usado como um monitor do sistema que mostra a atividade do processador em tempo real, exibindo as tarefas que estão sendo executadas na CPU e fornecendo uma interface amigável para o gerenciamento de processos?

- ☐ grep
- ☐ mkdir
- ☐ os
- ☐ top
- ☐ egrep

Exercício de fixação 3
Visualização de árvore de processos

Qual comando é usado para visualizar a árvore de processos?

- ☐ apple
- ☐ tree
- ☐ pstree
- ☐ bg
- ☐ fg

Shell Script

Shell Script é uma linguagem de programação baseada no conceito de interpretação, que pode ser utilizada na linha de comando do Shell. Os programas escritos nessa linguagem são chamados de scripts.

- Com os comandos separados por ";":

```
# cd /home/andreia/backup/; tar cvf  bkk.tar /var/log/*.log
```

Ou pode ser utilizada para automatizar sequências de comandos em arquivos executáveis especiais.

- Com os comandos separados por ";"ou colocados um em cada linha:

```
# file arquivo   # utilizado para ver se um arquivo é script.
```

Com o objetivo de automatizar sequências de tarefas que serão repetidas várias vezes, encapsulamo-nas em arquivos executáveis especiais. Neles podemos escrever sequências de comandos do sistema operacional, tal como fazemos no Shell, adicionando também a lógica de programação.

Por exemplo, uma das atividades rotineiras de um administrador de sistemas Linux é realizar operações de backup, mantendo cópias de segurança dos arquivos importantes. Supondo que vamos fazer um backup bem simples de, por exemplo, arquivos *.log* gerados pasta */var/log/*, e compacta-los, gerando um arquivo com extensão *.tar* , colocando-o em um diretório chamado backup, na área de usuário. Uma sequência possível de comandos a ser digitada no Shell seria a seguinte:

```
# cd /home/usuario/backup

# tar cvf  bk.tar /var/log/*.log
```

Supõe-se aqui que o usuário atual seja chamado de “usuario” e que exista o diretório backup já previamente criado.

Se tal atividade for realizada periodicamente, é interessante que seja criado um programa para automatizar essa tarefa. No Shell, os programas são interpretados, e por essa característica são chamados de scripts. Assim, os scripts nada mais são do que programas contendo sequências de comandos que são interpretados pelo Shell, linha após linha. O script é um arquivo executável, com diretrizes na linha inicial que diz qual Shell deverá interpretar aquela sequência de comandos quando o arquivo for executado. Essa linha deve ser a primeira linha do script e começa com os caracteres #! (chamados de *shebang*) seguidos do caminho na árvore de diretórios no qual o Shell será encontrado e qual será ele.

O sistema de arquivos do Linux identifica um script através do conteúdo dos seus dois primeiros bytes. No entanto, nem todos os arquivos são identificados através do conteúdo dos seus dois primeiros bytes. Para verificar o tipo de um arquivo, pode-se utilizar o comando *file*, cuja sintaxe é a seguinte:

```
# file arquivo
```

Para indicar o Shell *bash* como o interpretador daquela sequência de comandos contida no arquivo, supondo que o interpretador *bash* esteja no diretório */bin*, devemos ter a primeira linha como:

```
#!/bin/bash
```

Podemos então, criar um arquivo chamando de, por exemplo, “scriptBackup”, e nele colocar os comandos necessários para realizar o backup dos arquivos:

```
#!/bin/bash

cd   /home/usuario/backup              # aqui você coloca o
comentário

 tar cvf  bk.tar /var/log/*.log
```

Por padrão, os comandos são colocados um em cada linha. Para colocá-los na mesma linha basta separá-los com “;”. Para colocar linhas que não devem ser interpretadas, ou seja, acrescentar comentários no script, basta introduzir o caractere # antes da frase.

Para executar a sequência de comandos do script basta executar o arquivo na linha de comando do Shell:

```
# ./scriptBackup
```

Lembre-se que o arquivo deve ser executável. Para tanto, devemos dar a permissão de “x” para o arquivo, usando, por exemplo, o comando *# chmod +x scriptBackup*.

Podemos ainda, incrementar o script, adicionando mensagens para que o usuário acompanhe o que está sendo feito e também, por exemplo, incrementar o nome do arquivo de backup com a data, usando o comando *date*.

```
#!/bin/bash

echo “Realizando o backup dos arquivos de Log”

cd /home/usuario/backup
```

```
tar cvf  bk`date +%d%m%Y`.tar /var/log/*.log

      echo “Backup concluído”
```

O comando *date +%d%m%Y* trará a data atual do sistema no formato dia, mês e ano em quatro dígitos. Ele está entre “crases” para indicar ao Shell que ele deve ser interpretado e o resultado dele deve ser colocado naquele ponto. Assim se a data atual do sistema for 08/02/2012, o nome do arquivo a ser criado será *bk08022012.tar*.

Outro exemplo de script, cuja sequência de comandos busca por informações do usuário atual, pode ser:

```
#!/bin/bash

echo “Informações do usuário atual”

cat /etc/passwd | grep `whoami`
```

Temos, na linha 3, três comandos que devem ser executados nesta ordem. O comando *cat* listará o arquivo *passwd* que contém informações sobre todos os usuários do sistema; o resultado do comando servirá de entrada para o comando *grep*, que tem por propósito filtrar todas as linhas que contenham o padrão do parâmetro. No entanto este padrão só será conhecido após a execução do comando *whoami*, que busca qual o usuário atual. Por isso ele está entre “crases”.

Exercício de fixação 4
Criando um script simples

Crie um script para remover todos os arquivos texto do diretório */home* do usuário.

Se necessário, crie previamente alguns arquivos de texto no diretório.

Não se esqueça de alterar o script para que ele se torne executável (*#chmod +x script)*.

Variáveis do Shell Script

São utilizadas para manipular as posições de armazenamento na memória do computador. É possível acessar essa posição armazenando, lendo e modificando o dado nela gravado. Uma variável tem um nome e um valor associado a ela. Os conteúdos das variáveis são preenchidos através do comando de atribuição “=”.

Não deve haver espaço entre os elementos de uma atribuição.

```
# teste=10     #
```

Para visualizar o conteúdo de uma variável:

```
# echo $teste
```

Outros exemplos:

```
# X=1                    #funciona
# X=teste              #funciona
# X=varias palavras          #erro
# X=”varias palavras”      #funciona
# X=”$Y unidades”              # o $Y (será substituído pelo
  valor da variável Y)
# X=’$Y unidades’         # $Y (não será substituído)
```

As variáveis são usadas para manipular as posições de armazenamento na memória do computador. Assim, através da variável é possível acessar essa posição, armazenando, lendo, modificando o dado lá gravado. Uma variável tem um nome e um valor associado a ela. Assim, por exemplo, podemos ter uma variável de nome “teste” com o valor 10 como seu dado ou conteúdo.

Os conteúdos das variáveis são preenchidos de duas formas. Uma delas é através de um comando de atribuição (=). Por exemplo:

```
# teste=10               # não deve haver espaço entre os elementos
de uma atribuição
```

Para visualizar o conteúdo de uma variável, podemos usar o comando *echo*:

```
# echo $teste
```

Para indicar ao Shell que aquela letra ou sequência de caracteres deve ser interpretada como uma variável, basta usá-la ao longo do programa.

O nome de uma variável é atribuído pelo programador. Por padrão, o nome deve ser representativo do que ela vai armazenar. No entanto, não se pode nomear uma variável com nomes já usados em comandos ou itens da sintaxe da linguagem script. Os nomes de variáveis começam sempre com uma letra e são seguidos de um ou mais números e letras. Não são permitidos caracteres especiais para construir o nome de uma variável, apenas o caractere *underline* (“_”), como por exemplo, soma_total.

Apesar de não ser necessário, por ser característica própria das linguagens scripts, as variáveis podem ser declaradas antes de serem usadas. Isso é prática em programas grandes, com uma grande quantidade de variáveis. A declaração se dá com o comando *declare* e acontece de praxe no início do programa. Assim por exemplo podemos criar uma variável, atribuir um valor a ela e mandar mostrar na tela, como no exemplo abaixo:

```
#!/bin/bash
# este script declara uma variável
declare curso
curso=”shell script”
      echo “Estamos estudando o conteúdo de $curso”
```

Alguns outros exemplos de atribuições:

```
# X=1 #funciona

# X=teste #funciona

# X=varias palavras #erro

# X=”varias palavras” #funciona

# X=”$Y unidades”     # o $Y (será substituído pelo valor da
variável Y)

# X=’$Y unidades’      # $Y (não será substituído)
```

Um recurso poderoso do bash também pode ser usado na atribuição a uma variável: a utilização da saída de um comando, que pode ser feita de duas formas, utilizando `<comando>` (*back ticks* ou crases), ou a forma $(<comando>), conforme o exemplo:

```
# Y=$(date)

# Z=`uptime`
```

Um cuidado deve ser tomado com as formas citadas acima: o bash não preserva a formatação da saída destes comandos, eliminando, assim, tabulações e quebras de linhas.

Escopo das variáveis

É chamado de escopo o espaço na qual uma variável é visível e pode ser utilizada.

Uma variável criada dentro de um script só é visível e manipulável dentro desse script.

- Chamada de variável local: todas as variáveis criadas nos scripts.

```
# X=1
```

Outras variáveis podem se tornar visíveis dentro e fora de um script, visíveis em todo o ambiente de um Shell.

- Chamadas de variáveis de ambiente.
- Criadas através do comando *export*:

```
# export X

# export Y=”/home/usuario/teste”
```

Chamamos de escopo de uma variável o espaço onde ela se torna visível e pode ser utilizada. Uma variável criada dentro de um script só é visível e manipulável dentro deste script; outras variáveis podem se tornar visíveis dentro e fora de um script, visíveis em todo o ambiente de um Shell. Ao primeiro tipo de variável chamamos de “variáveis locais” e, ao segundo, chamamos de “variáveis de ambiente”. Para criar uma variável local, basta lhe atribuir um valor. Porém, para criar uma variável de ambiente que será utilizada no Shell corrente e em todos os scripts ou sub-shells lançados a partir do Shell atual, é necessário usar o comando *export*. Este comando pode ser utilizado para transformar uma variável local em global, ou para criar uma variável local, conforme nos exemplos abaixo:

```
# X=1

# export X

# export Y=”/home/usuario/teste”
```

Ao iniciar um Shell, é provável que muitas variáveis de ambiente já estejam definidas. Para listá-las, pode-se utilizar os comandos *env* ou *printenv*. Para listar todas as variáveis (locais e de ambiente), utilize o comando *set* sem argumentos.

Expressões e testes

1º formato:

```
((expressão))
```

Usado quando será avaliada uma operação aritmética.

Por exemplo, para fazer com que uma variável y receba o conteúdo da soma 5 + 6.

```
((y=5 + 6))
```

As expressões em Shell são usadas em três formatos:

```
((expressão))
```

Usada quando será avaliada uma operação aritmética. Por exemplo, para fazer com que uma variável y receba o conteúdo da soma 5 + 6, fazemos:

```
((y=5 + 6))
```

Assim, são suportadas várias operações aritméticas, tal como mostrado na Tabela 6.3.

Operação	Sinal	Exemplo
Adição	+	res=$((1 + 1)) ou ((res= 1+1))
Subtração	-	res=$((4 - 1)) ou ((res=4-1))
Multiplicação	*	res=$((3 * 6)) ou ((res=3*6))
Exponenciação	**	res=$((2**3)) ou ((res=2**3)) #resultado=8
Divisão de inteiros	/	res=$((5 / 2)) ou ((res=5/2) #resultado=2
Resto	%	S=$((5 % 2)) ou ((res=5%2)) # resultado=1
Deslocamentos de bits à esquerda	<<	S=$((5 << 2)) #desloca 2 bits à esquerda, resultado=20
Deslocamentos de bits à direita	>>	S=$((4 >> 2)) #resultado=2

Tabela 6.3
Operações aritméticas.

Há ainda os operadores de incremento (++) e decremento (--), que somam, ou subtraem de uma unidade. Assim, se tivermos uma variável x com valor 2 e aplicarmos a operação ((x++)), teremos x com valor 3.

As precedências pelos operadores aritméticos seguem a ordem descendente:

- ++ -- ** * / % + -

Por exemplo, na expressão ((r=3 – 2 * 4)) o operador * será executado primeiro. Assim, o valor de r será -5. Se quisermos mudar a precedência, basta utilizarmos parênteses:

((r=(3-2)*4)), cujo resultado será 4.

São suportadas também operações de comparação aritmética:

Operação	Sinal	Exemplo
Igualdade	= =	(($X = = $Y))
Desigualdade	! =	(($X ! = $Y))
Maior que	>	((5 > 3))
Menor que	<	((2 < 3))
Maior ou igual	>=	(($X >= 3))
Menor ou igual	<=	(($Y <= 1))

Tabela 6.4
Operações de comparação aritmética.

Note que resultado de uma operação de comparação será 0 (zero) ou 1 (um), dependendo da comparação resultar em falso ou verdadeiro. Exemplo:

```
# echo $((5 ==5))                    # resultado: 1

# echo $(( 5 > 10 ))                 #resultado: 0
```

Com relação à precedência, os operadores de comparação aparecem após os aritméticos, na ordem:

```
    <=    >=    <     >              ==    ! =
```

O bash suporta, ainda, as principais operações lógicas, como E, OU e a negação. Estes operadores também podem ser combinados na avaliação de condições com vários termos.

Operação	Sinal	Exemplo
E lógico	&&	(condição1) && (condição2)
OU lógico	||	(condição1) || (condição2)
Negação (NOT)	!	!(condição1)

Tabela 6.5
Principais operações lógicas.

Expressões e testes (2º formato):

```
[expressão] ou test expressão
```

Usados para fazer testes em números, textos e arquivos.

Este tipo de formato também é capaz de fazer testes em números, textos e arquivos. Outra forma de fazer isso é usando o comando *test*. Algumas opções de parâmetro para uso são mostradas na Tabela 6.6:

Operação	Sinal	Exemplo
Igualdade	- eq	[5 -eq 5]
Desigualdade	- ne	[5 -ne 4]
Maior que	- gt	[6 -gt 5]
Menor que	- lt	[5 -lt 6]
Maior ou igual	- ge	[6 -ge 5]
Menor ou igual	- le	[5 -le 6]

Tabela 6.6
Opções de parâmetro.

Além de ser usado com variáveis/números, o comando *test* também faz operações com arquivos. Por exemplo, para saber se o conteúdo da variável de ambiente HOME é um diretório, podemos usar o comando:

```
# test -d "$HOME" ; echo $?
```

O comando *echo* escreve o valor 0 como o conteúdo da variável de status, pois o comando foi bem-sucedido, ou seja, $HOME tem um diretório como conteúdo. E combinações:

```
# test "$HOME" != "/usr"  -a 5 -le 7; echo $?
# [ "$HOME" != "/usr"  -a 5 -le 7 ]; echo $?                    #
forma alternativa
```

Expressões e testes (3º formato):

```
 [[expressão]]
```

Esse formato é usado para testar atributos de um arquivo ou diretório, fazer comparações com strings e algumas comparações numéricas. Exemplo:

```
[[ (-d "$HOME") && (-w "$HOME") ]] ; echo $?
```

Conforme vimos, uma sequência de comandos pode vir na mesma linha desde que separada por ";". No exemplo acima, usamos o comando *echo* para mostrar o conteúdo do parâmetro *?*, que guarda o status de execução do último comando executado; se zero, o comando foi bem-sucedido; caso contrário, não. Assim, como o conteúdo de HOME é um diretório, ou seja, o comando foi bem-sucedido, o comando *echo* deve imprimir valor zero. Se usássemos a opção *–f*, que verifica se é um arquivo, o resultado seria diferente de zero.

Outras opções podem ser observadas na Tabela 6.7.

Tabela 6.7
Outras opções.

Característica		Característica	
-d	É um diretório	-f	É um arquivo normal
-e	Se o arquivo existe (também usado -a)	-r	Tem permissão de leitura para meu usuário
-n	Se o arquivo não está vazio	-w	Tem permissão de leitura para meu usuário
-N	Foi modificado desde a última leitura		

Para testes entre pares de arquivos:

Operaçao	
-nt	Testa se o arquivo 1 é mais novo que o arquivo 2, considerando a data
-ot	Testa se arquivo 1 é mais velho que arquivo 2, considerando a data

Tabela 6.8
Testes em pares de arquivos.

As opções *–a* e *–o* permitem combinar expressões com operadores lógicos E e OU, respectivamente. Podemos inclusive combinar as opções:

```
# test "$HOME" != "/usr"  -a 5 -le 7; echo $?
# [ "$HOME" != "/usr"  -a 5 -le 7 ]; echo $?        # forma alternativa
```

Na qual temos uma sequência de testes onde verifica-se se o conteúdo da variável $HOME é diferente (!=) de /usr. No caso, sim. Ou seja, o resultado é 0. O próximo teste é: 5 é menor que (-le) 7? Sim, o resultado é 0. O parâmetro *–a*, quando usado com expressões, significa que deve ser feita uma operação lógica E entre o resultado do que vem antes dele com o

resultado do que vem depois dele. A operação E só retorna verdadeiro se ambos forem verdadeiros. Com o resultado dos dois lados é zero, o resultado da operação também será zero, ou seja, tudo verdade no teste.

```
[[ expressão ]]
```

Esse formato é usado para testar atributos de um arquivo ou diretório, fazer comparações com strings e algumas comparações numéricas. Por exemplo:

```
[[ (-d "$HOME") && (-w "$HOME") ]] ; echo $?
```

A variável HOME contém um diretório e este diretório tem permissão de escrita, o resultado será zero no comando *echo*, ou seja, bem sucedido.

Comando *read*

- Lê um valor da entrada padrão e coloca o conteúdo na variável indicada.
- A utilização do comando *read* é outra forma de atribuir valor a uma variável.
- O comando *read* pode também acumular a função que seria do comando *echo*, de mostrar algo na tela.
- Através do parâmetro *–p* é possível combinar saída e entrada no mesmo comando *read*.

Outra forma de atribuir valor a uma variável é através do comando *read*. Quando acionado o comando *read* lê um valor da entrada padrão e o coloca como conteúdo da variável indicada. Por exemplo:

```
#!/bin/bash
echo "Digite seu nome"
read nome
echo "Bom dia "$nome
```

No exemplo acima, o comando *echo* solicita ao usuário que ele digite o nome. O comando *read* fica a espera de que algo seja digitado e seja teclado *enter*. Quando isso acontece, atribui aquele conteúdo à variável *nome*. O comando *echo* então escreve esse conteúdo.

O comando *read* pode também acumular a função que seria do comando *echo*, de mostrar algo na tela. Através do parâmetro *–p* é possível combinar saída e entrada no mesmo comando *read*. Podemos transformar a sequência acima para esta, que faz a mesma ação:

```
#!/bin/bash
read -p "Digite seu nome: " nome
echo "Bom dia "$nome
```

Usado com o parâmetro -s o comando *read* oculta a entrada dos caracteres. Isso serve para, por exemplo, manipular senha.

```
#!/bin/bash
read -p "Usuário: " usuario
read -p "Senha: " -s senha
echo -e "\n O seu usuário é $usuario e sua senha é $senha"
```

- A opção –*e* no comando *echo* faz com que seja interpretada a mudança de linha (\n).
- Um mesmo comando *read* pode atribuir valores a diversas variáveis. Qualquer entrada separada por espaço em branco caracteriza-se como diversos valores.

```
# read valor1 valor2
```

- Podemos ainda adicionar testes para incrementar um pouco nosso script, usando o comando *test*.

```
#!/bin/bash

echo “Você está entrando em uma área segura. Deseja continuar?
[sn]”

read resposta

test “$resposta” = “n” && exit

read -p “Usuário: “ usuario

read -p “Senha: “ -s senha

echo -e “\n O seu usuário é $usuario e sua senha é $senha”
```

Forma alternativa do comando:

```
test “$resposta” = “n” à    [ $resposta = n ]
```

Neste exemplo, o comando *test* verifica se o valor da variável resposta é “n”; se for, aplica o comando *exit*, que encerra a execução do script; senão, continua a executar o script.

No exemplo acima podemos substituir o comando *test “$resposta” = “n”* por *[$resposta = n]*. O resultado será o mesmo.

Parâmetros de linha de comando (variáveis especiais)

- Um parâmetro é um nome, número ou caractere especial que armazena um valor.
- Parâmetro de linha de comando é chamado de parâmetro posicional.
- São usados para passar valores junto com o nome do programa na linha de comando.

Esses valores são armazenados nos parâmetros especiais e podem ser usados dentro do script.

```
# ./soma 3 9
```

Ao interpretar essa linha de comando, o Shell atribui os valores ali encontrados nos parâmetros posicionais, começando do valor 0 e indo até o valor 9.

- Parâmetro 0 – nome do programa.
- Parâmetro 1 – o número 3.
- Parâmetro 2 – o número 9.

Um parâmetro é um nome, número ou caractere especial que armazena um valor. Um parâmetro de linha de comando é chamado de parâmetro posicional. São usados para passar valores junto com o nome do programa na linha de comando. Esses valores são armazenados nos parâmetros especiais e podem ser usados dentro do script. Assim, para elaborar

um script que se comporta como uma calculadora, somando quaisquer dois números, podemos deixar que o usuário de nosso script diga quais são esses dois números quando for executar o script:

```
# ./soma 3 9
```

- Assim, soma seria o script, 3 e 9 seriam os números a serem somados.
- Ao interpretar essa linha de comando, o Shell atribui os valores ali encontrados nos parâmetros posicionais, começando do valor 0 e indo até o valor 9. Assim, o primeiro valor, que é o nome do programa é armazenado no parâmetro 0, o número 3, no parâmetro 1 e o 9 no parâmetro 2. Para acessar o conteúdo de cada parâmetro basta, por exemplo, usa-lo com o $.

```
#!/bin/bash

echo “o nome do programa passado no parâmetro 0 é: “ $0

echo “o parâmetro 1 recebeu o valor: “ $1

echo “o parâmetro 2 recebeu o valor: “ $2
```

Para acessar o conteúdo de cada parâmetro basta, por exemplo, usá-lo com o “$”.

- Apesar de existirem apenas 9 parâmetros posicionais, é possível passar quantos valores forem necessários na linha de comando.
- Esses valores ficam armazenados numa pilha. Os nove primeiros valores são alocados nos parâmetros.
- Para acessar os valores que ficaram na pilha, basta usar o somando shift “quantidade”.
- Assim, shift 1 diz ao shell para que seja ignorado o primeiro valor e que os demais sejam deslocados em uma posição.
- O parâmetro 1 passa a ter o segundo valor, e o nono parâmetro passa a ter o 10º valor. Esse deslocamento pode ser de qualquer valor, ditado pelo comando *shift*.
- Alguns parâmetros especiais são também usados pelo Shell para nos mostrar algumas informações. Por exemplo, o parâmetro # traz a quantidade de parâmetros recebidos em linha de comando. Se quisermos saber quantos parâmetros basta mandar escrever $#. Há ainda o parâmetro @ que contem uma lista de todos os parâmetros recebidos. Observe no exemplo abaixo:

```
#!/bin/bash

echo “o nome do programa passado no parâmetro 0 é: “ $0

echo “o parâmetro 1 recebeu o valor: “ $1

echo “o parâmetro 2 recebeu o valor: “ $2

echo “o total de parâmetros recebidos é: “  $#

echo “lista de todos os parâmetros recebidos: “  $@
```

Outros parâmetros também são usados pelo Shell e que podemos visualizar:

Parâmetro	Descrição
?	Contém o status de saída do último comando a ser executado no Shell. Se o comando foi executado com sucesso, tem valor 0; se não, terá valor diferente de 0.
$	Contém o PID do shell.
!	Contém o PID do último processo executado em background.

Tabela 6.9
Parâmetros do Shell.

Roteiro de Atividades 6

Atividade 6.1 – Exibindo processos em estados específicos

Pesquise as opções do comando *ps* e crie um comando para listar apenas os identificadores e nomes dos processos que estão no estado R (running).

Verifique também o comando *top,* uma versão interativa do comando *ps*, atualizando a listagem de processos a cada *n* segundos e ordenando-os por uso de CPU e memória.

Atividade 6.2 – Executando processos em background

A partir do Shell, execute um programa em background (por exemplo, o navegador web). Observe a árvore hierárquica de processos e visualize as dependências destes processos. Observe ainda seu PID e PPID e compare aos do bash.

Verifique também o comando *jobs*. Ele mostra a situação de todos os processos que estão em background naquele Shell.

Atividade 6.3 – Utilizando um daemon

Utilize o daemon *cron* para realizar um backup daqui a cinco minutos dos arquivos *.log* da pasta */var/log*. O arquivo de backup deve ser compactado e colocado dentro de sua pasta.

Utilize *crontab –e* para editar o arquivo e o comando *tar –zcvf destino origem* para fazer a compactação dos arquivos *.log*.

Atividade 6.4 – Usando testes dentro dos scripts

Crie um script que mostra qual é o usuário atual, qual o diretório atual e o Shell que está usando.

Uma dica é lembrar que no arquivo */etc/passwd* encontram-se listados os usuários e seus shells.

Use o comando *cut* com a sintaxe:

```
cut -d DELIMITADOR -f NUMERO_DO_CAMPO
```

O comando cut corta somente o(s) campo(s) de número NUMERO_DO_CAMPO e utiliza o separador DELIMITADOR para delimitar cada campo. No caso do arquivo *passwd*, o delimitador é ":".

Atividade 6.6 – Lendo variáveis e usando expressões nos scripts

Complete o script que lê três números, armazena-os em três variáveis (usando o comando *read*) e escreve na tela a soma e a subtração deles.

#!/bin/bash

echo "Digite o primeiro número"

_____

echo "Digite o segundo número"

_____

echo “Digite o terceiro número”

_____

_____

_____

echo “ A soma destes números é $soma”

echo “ A subtração destes números é $sub”

Atividade 6.7 – Utilizando parâmetros

Crie um script que recebe 12 números por parâmetro e em seguida escreve na tela quais são eles. Como são apenas 9 os parâmetros mapeados ($1 até $9) você deverá usar o comando *shift 3* para deslocar os três valores restantes.

Atividade 6.8 – Utilizando parâmetros e testes

Crie um script que verifica o número de parâmetros recebidos na linha de comando. Para esta atividade, você deve verificar se existe apenas um parâmetro. Use o comando *test* para verificar e o comando *exit* para sair, se for número incorreto de parâmetros. Lembre-se de que o número de parâmetros está na variável *$#*.

Atividade 6.9 – Utilizando testes de diretório

Continue a atividade anterior para verificar se o parâmetro é um diretório. Mande uma mensagem dizendo ao usuário se é ou não.

Exemplo: *./testedir /home*

Use o teste *[[(-d $1)]]* para verificar isso. Lembre-se que a variável *$?* traz o status da execução; se zero, o comando foi bem-sucedido; se diferente de zero, o comando não foi bem-sucedido; no caso, o parâmetro recebido não é o nome de um diretório.

Atividade 6.10 – Listando arquivos passados por parâmetro

Continue a atividade anterior agora para receber dois parâmetros: um nome de diretório (tal como na anterior) e uma letra. Se o parâmetro 1 for um diretório, o script deve listar todos os arquivos que começam com a letra recebida no parâmetro 2.

Use o comando *ls $2**.

7

Shell Script

objetivos

Aprender a controlar o fluxo de execução sequencial num script com o uso de estruturas de decisão e a criar máscaras ou padrões de pesquisas na busca por informações através de expressões regulares.

conceitos

Estruturas de decisão e expressões regulares.

Estruturas de decisão

- São usadas para direcionar o fluxo de execução do programa.
- As ações só são tomadas após a verificação de uma condição.
  - Para cada alternativa, um conjunto diferente de comandos é executado.

Esse conceito é implementado no Shell através de duas estruturas:

- Com o comando *if*.
- Ou com o comando *case.*

Uma decisão é tomada a partir de possíveis alternativas baseadas em alguma condição. Ou seja, antes de executar uma ação, é verificada uma condição. Este conceito é implementado no Shell através de duas estruturas: o comando *if* e o comando *case*.

Comando *if*

Sintaxe do comando:

```
if <condição>
then
   <comandos>
fi
```

Semântica do comando:

- Avalia-se a condição.
- Se, após avaliada a condição, o valor retornado for:
  - **Verdadeiro**: executam-se os comandos dentro do *if*.
  - **Falso**: não se executam os comandos.

Durante a execução de um script, pode ser necessário executar um grupo de comandos somente mediante a observação de um determinado cenário ou condição. Caso não seja, este grupo de comandos não deverá ser executado. O bash suporta as operações de execução condicional por meio do comando *if*, que possui a seguinte forma:

```
if <condição>
then
   <comandos>
fi
```

Se a condição, após ser avaliada, retornar "falso", os comandos do *if* não serão executados; se a avaliação gera um valor "verdadeiro", os comando dentro do *if* serão executados.

A condição pode ser o resultado de um comando:

```
#!/bin/bash
if ls /home
then
   echo "Este é o conteúdo da pasta home"
fi
```

- No caso, a condição do *if* é o retorno do comando *ls /home*.
- Se existir a pasta *home*, ele a lista e retorna comando bem-sucedido, fazendo com que a condição do *if* seja verdadeira. Assim, o comando *echo* é executado.

A condição pode ser também o resultado de uma comparação aritmética. Por exemplo:

```
#!/bin/bash
read -p "Digite um número: " a
read -p "Digite outro número: " b
if (( a + b > 3 ))
then
   echo "A soma dos dois valores é maior que 3"
fi
```

Neste script são lidos dois valores e, a partir daí, executada a expressão da condição (a + b > 3). Se a expressão for avaliada como verdadeira, o comando *echo* será executado.

A condição pode ainda ser qualquer padrão de expressão e teste, visto na unidade anterior. O exemplo abaixo testa se o arquivo *bash* existe no diretório *bin*:

```
#!/bin/bash
if [ -a /bin/bash]
then
echo "o arquivo existe"
fi
```

Comando *if* – tipos de condição

A condição do comando *if* pode ser o resultado de um comando.

```
#!/bin/bash
if ls /home
then
   echo “Este é o conteúdo da pasta home”
fi
```

A condição do *if* é o retorno do comando *ls /home*.

- Se existir a pasta “home”, ele a lista e retorna comando bem-sucedido, fazendo com que a condição do “if” seja verdadeira e, por consequência, executando o comando *echo*.

Comando *if ... else*

Sintaxe do comando:

```
if <condição>
then
<comandos  executados caso a condição seja verdadeira>
else
<comandos  executados caso a condição seja falsa>
fi
```

Semântica do comando:

- Se a condição for avaliada como verdadeira: executam-se os comandos do *if.*
- Se a condição for avaliada como falsa: executam-se os comandos do *else*.

A forma básica do comando *if* permite a execução de um grupo de comandos, caso uma condição ou grupo de condições sejam verdadeiras. No entanto, é bastante comum ser necessário executar um grupo de comandos quando uma condição for verdadeira e outro grupo de comandos quando a condição for falsa. Para tanto, são utilizados os comandos *if ... else*, com essa sintaxe:

```
if <condição>
then
<comandos  executados caso a condição seja verdadeira>
else
<comandos  executados caso a condição seja falsa>
fi
```

Por exemplo:

```
#!/bin/bash
echo
read -p “Digite um número: “ num
```

```
if [ $num -lt 10 ]
then
   echo “O número $num é menor que 10”
else
   echo “O número $num é maior ou igual a 10”
fi
```

Neste script é lido um número. Se esse número for menor que 10 (-lt), ou seja, a avaliação da condição retornar um valor verdadeiro, executa-se o primeiro comando *echo*. Caso contrário, executa-se o segundo comando *echo*, aquele que está sob o *else*.

Outra forma de utilização do comando *if* é quando se deseja “aninhar” outras comparações: caso uma condição seja falsa, executa-se outro *if.* Para tanto, é utilizado o comando *elif* no lugar do *else*, da seguinte forma:

```
if <condição>
then
  <comandos executados caso a condição seja verdadeira>
elif <outra condição>
then
  <comandos executados caso a outra condição seja verdadeira>
else
  <comandos executados caso nenhuma opção seja verdadeira>
fi
```

Por exemplo:

```
#!/bin/bash
read -p “Digite um número: “ num
if [ $num -lt 10 ]
then
   echo “O número $num é menor que 10”
elif [ $num -gt 10 ]
then
   echo “O número $num é maior que 10”
else
   echo “O número é igual a 10”
fi
```

Este script desmembra o script que de verificação anterior, agora apontando se o número é menor, maior ou igual ao número 10.

Exemplo:

```
#!/bin/bash
read -p “Digite um número: “ num
if [ $num -lt 10 ]
then
   echo “O número $num é menor que 10”
else
   echo “O número $num é maior ou igual a 10”
fi
```

- Lê-se um número (read).
- Se esse número for menor que 10 (-lt), executa-se o primeiro comando *echo*.
- Caso contrário, executa-se o segundo comando *echo*, aquele que está sob o *else*.

Aninhando comandos ‘if’ e ‘else’

Sintaxe do comando:

```
if <condição>
then
  <comandos executados caso a condição seja verdadeira>
elif <outra condição>
then
  <comandos executados caso a outra condição seja verdadeira>
else
  <comandos executados caso nenhuma opção seja verdadeira>
fi
```

Semântica do comando:

- Se a condição do *if* for verdadeira, executam-se os comandos dele.
- Caso contrário, avalia-se a condição do *elif*.
  - Se for verdadeira: executam-se os comandos dele.
  - Caso contrário: executam-se os comandos do *else*.

Exemplo:

```
#!/bin/bash
read -p “Digite um número: “ num
if [ $num -lt 10 ]
then
   echo “O número $num é menor que 10”
```

```
elif [ $num -gt 10 ]
then
   echo “O número $num é maior que 10”
else
   echo “O número é igual a 10”
fi
```

Esse script desmembra o script de verificação anterior, agora apontando se o número é menor, maior ou igual ao número 10.

Exercício de fixação 1
Estrutura de decisão

Complete o script abaixo para que receba como parâmetro dois números e verifique qual é o maior.

#!/bin/bash

_____ [_____ > _____]

_____

echo “ O número $1 é maior que o número $2”

_____

echo “O número $2 é maior que o número $1”

_____

Comando *case*

Sintaxe do comando:

```
case $variavel in
  valor1) <comandos>
  ;;
  valor2) <comandos>
  ;;
  valor) <comandos>
  ;;
esac
```

Semântica do comando:

- O conteúdo da variável é comparado aos valores “valor1”, “valor2” e “valor3”.
  - São executados os comandos sob o valor que se encaixa (aquele que valida o teste).

Caracteres-Curinga:

- *
  - Zero ou mais caracteres quaisquer.

- ?
  - Um caractere qualquer.

Formam um padrão de casamento.

Exemplo:

- o padrão “st*” casaria com valores tais como “start”, “stop”, “st” ou “stabcd”
- Já o padrão “k?” casaria apenas com valores que iniciassem com a letra “k” e tivessem exatamente mais um caractere, como, por exemplo, os valores “k1” e “kO”.

No comando *case*, o “* ” pode ser usado como forma de *else* ou valor default.

- Se o valor da variável não casou com nenhum outro padrão anterior, casará com o “*”.

Exemplo:

```
case $valor in

start) echo “iniciando”

;;

stop) echo “parando”

;;

*)echo “valor não encontrado, use apenas start ou stop”

;;

esac
```

O comando *case* provê outra forma de execução condicional, sendo apropriado quando for necessário executar um comando de acordo com uma opção (tipicamente, um valor contido em uma variável). A sintaxe do comando é:

```
case $variavel in

  valor1) <comandos>

  ;;

  valor2) <comandos>

  ;;

  valor) <comandos>

  ;;

esac
```

No exemplo, o conteúdo da variável é comparado aos valores “valor1”, “valor2” e “valor3”. Além de valores exatos como os usados no exemplo acima, podemos utilizar também caracteres curinga, como o * e ? para formar um padrão de casamento. Por exemplo, o padrão st* casaria com valores tais como “start”, “stop”, “st” ou “stabcd” (“*” indica zero ou mais caracteres quaisquer). Já o padrão “k?” casaria apenas com valores que iniciassem com a letra k e tivessem exatamente mais um caractere como, por exemplo, os valores k1 e kO (*?* indica um caractere qualquer).

Uma técnica utilizada em blocos case é colocar, como último padrão, o * como uma forma de *else*, ou seja, se o valor da variável não casou com nenhum outro padrão anterior, casará com o *, conforme mostra o exemplo abaixo:

```
case $valor in

start) echo “iniciando”

;;

stop) echo “parando”

;;

*)echo “valor não encontrado, use apenas start ou stop”

;;

esac
```

Lembre-se de que cada bloco de comandos é separado do padrão seguinte por “;;” e que o comando *case* é encerrado com um *esac*.

Expressões regulares

As expressões regulares são usadas para descrever formalmente os padrões de texto.

Com o seu uso é possível criar uma máscara ou padrão de pesquisa para buscar informações.

- As pesquisas se tornam mais abrangentes e mais poderosas.

Os padrões são descritos por meio de metacaracteres.

- Se a pesquisa pelo padrão gera um resultado positivo, diz-se que o texto “casou” com a expressão.
- Considerando um extrato do arquivo */etc/passwd*.
- Usando o comando *egrep*.

```
root:x:0:0:root:/root:/bin/bash

daemon:x:1:1:daemon:/usr/sbin:/bin/sh

bin:x:2:2:bin:/bin:/bin/sh

couchdb:x:113:124:CouchAdministrator,,,:/var/lib/couchdb:/bin/bash

lightdm:x:114:125:Light Display Manager:/var/lib/lightdm:/bin/false

colord:x:115:126:comanagement daemon,,,:/var/lib/colord:/bin/false

usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash

usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash
```

As expressões regulares são usadas para descrever formalmente os padrões de texto. Com seu uso é possível criar uma máscara ou padrão de pesquisa para buscar informações. Assim, as pesquisas se tornam mais abrangentes e mais poderosas.

Os padrões são descritos através de metacaracteres. Se a pesquisa pelo padrão gera um resultado positivo, diz-se que o texto "casou" com a expressão. Nas Tabelas 7.1 a 7.4 são abordados alguns deles. Para exemplificar cada um desses elementos, tomamos como base o arquivo */etc/passwd*. Em um extrato feito como exemplo, consideraremos este conjunto de linhas:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
couchdb:x:113:124:CouchAdministrator,,,:/var/lib/couchdb:/bin/bash
lightdm:x:114:125:Light Display Manager:/var/lib/lightdm:/bin/false
colord:x:115:126:comanagement daemon,,,:/var/lib/colord:/bin/false
usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash
usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash
```

No qual os campos, separados por ":", são respectivamente: login de usuário, senha, UID, GID, comentários do usuário tal como nome, diretório home e shell do usuário.

Tabela 7.1 Descrição de padrões por metacaracteres.

Para exemplificar cada um dos casos do uso das expressões, pegamos emprestado o comando *egrep*, que tem suporte nativo para expressões regulares, recuperando linhas no arquivo que seguem ou "casam" com aquele padrão.

Metacaractere	**Significado**
^	Indica o começo da linha
$	Indica o final da linha
Exemplo	**Semântica do comando**
# egrep ^u etc/passwd	*# todas as linhas que começam com a letra u* # **resultado** usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash
# egrep bash$ etc/passwd	*# todas as linhas que terminam com a palavra bash* **# resultado** root:x:0:0:root:/root:/bin/bash couchdb:x:113:124:CouchAdministrator,,,:/var/lib/couchdb:/bin/bash usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash

Listas	**Significado**
[abc]	Lista com três padrões: "a", "b" ou "c"
[a-d]	Lista de intervalos: procura por padrões no intervalo entre as letras, ou seja "a", "b", "c", "d"
[^abc]	Lista negada: procura por padrões que não são "a", "b" ou "c"
(joao|jose)	Procura por cadeias de caracteres "joao" ou (|) "jose"

Exemplo	Semântica do comando
#egrep Jo[as] etc/passwd	# busca pelos padrões que começam com Jo e em seguida vem a ou s # **resultado** usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash
#egrep ^[a-e] etc/passwd	# busca pelos padrões que começam a linha com as letras a,b,c,d ou e # **resultado** daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh couchdb:x:113:124:CouchAdministrator,,,:/var/lib/couchdb:/bin/bash colord:x:115:126:comanagement daemon,,,:/var/lib/colord:/bin/false
#egrep ^[aeiou] etc/passwd	# busca pelos padrões que começam a linha com as vogais # **resultado** usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash
#egrep ^[^aeiou] etc/passwd	# busca pelos padrões que começam a linha com as consoantes # **resultado** root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2:bin:/bin:/bin/sh couchdb:x:113:124:CouchAdministrator,,,:/var/lib/couchdb:/bin/bash lightdm:x:114:125:Light Display Manager:/var/lib/lightdm:/bin/false colord:x:115:126:comanagement daemon,,,:/var/lib/colord:/bin/false
#egrep '(Joao|Jose)' etc/passwd	# busca pelos padrões cujas linhas tenham a palavra Joao ou Jose # **resultado** usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash

Tabela 7.2
Descrição de padrões por meta-caracteres.

Quantidade	Significado
a{2}	Procura pela letra a aparecendo 2 vezes
a{2,4}	Procura pela letra a aparecendo de 2 a 4 vezes
a{2, }	Procura pela letra a no mínimo 2 vezes
a?	Procura pela letra a aparecendo zero ou uma vez
a*	Procura pela letra a aparecendo zero ou mais vezes
a+	Procura pela letra a uma ou mais vezes
Exemplo	**Semântica do comando**
# egrep '0{2}' /etc/passwd	*# busca por linhas contenham dois zeros seguidos* **# resultado** usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash

Exemplo	Semântica do comando
# egrep '0{1,4}' /etc/passwd	# busca por linhas que contenham de um a quatro zeros # **resultado** root:x:0:0:root:/root:/bin/bash usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash
# egrep 1+ /etc/passwd	*# busca por linhas que contenham um ou mais 1´s* # **resultado** daemon:x:1:1:daemon:/usr/sbin:/bin/sh couchdb:x:113:124:CouchAdministrator,,,:/var/lib/couchdb:/bin/bash lightdm:x:114:125:Light Display Manager:/var/lib/lightdm:/bin/false colord:x:115:126:comanagement daemon,,,:/var/lib/colord:/bin/false usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash

Tabela 7.3
Descrição de padrões por meta-caracteres.

Curingas	Significado
.	Casa com um caractere qualquer
.*	Casa com qualquer quantidade de caracteres
Exemplo	**Semântica do comando**
# egrep usuario. /etc/passwd	# busca pelos padrões cujas linhas tenham a palavra usuário seguida de qualquer caractere # **resultado** usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash
# egrep Jo.* /etc/passwd	#busca pelos padrões cujas linhas tenham Jo seguido de qualquer quantidade de caracteres # **resultado** usuario1:x:1000:1000:Joao da Silva,,,:/home/joao:/bin/bash usuario2:x:1001:1001:Jose da Silva,,,:/home/jose:/bin/bash

Tabela 7.4
Descrição de padrões por metacaracteres.

Operador '=~'

As expressões regulares podem ser combinadas da forma que for necessária para cumprir-se a tarefa desejada. Podem inclusive ser usadas nos scripts que criamos; basta que seja usado o operador de expressão regular: "=~", como por exemplo:

```
#!/bin/bash

senha="shellscript"

if [[  "$senha" =~ ^s.e.*t$  ]]

then

   echo "$senha é a senha correta"

else

   echo "$senha não é a senha correta"

fi
```

No código acima, a condição do comando if verifica se o conteúdo da variável senha começa com s, seguido por um caractere qualquer (.), seguido da letra e, seguido de qualquer número de caracteres, terminando com t ("$senha" =~ ^s.e.*t$). Este padrão casa com o valor "shellscript" da variável, e portanto escreve dizendo que a senha é correta.

Comando *sed*

O comando *sed* recebe como entrada o valor da variável senha (através do pipe "|") e substitui as letras "ll" por "LL", além de colocar o resultado dentro da própria variável senha.

Outros tipos de expressões regulares são ainda usados, como por exemplo, nos casos de substituição, com o uso de comando *sed*.

```
#!/bin/bash
senha="shellscript"
echo "senha antes: $senha"
senha=`echo $senha | sed 's/ll/LL/'`
echo "senha depois: $senha"
```

O comando *sed* recebe um arquivo ou um valor de uma entrada padrão e substitui nele o que está escrito na expressão. No exemplo acima, o comando *sed* recebe como entrada o valor da variável senha (através do pipe "|") e substitui as letras "ll" por "LL" e coloca o resultado dentro da própria variável senha.

Comando *tr*

Sua função é receber dois conjuntos de caracteres e substituir as ocorrências dos caracteres no primeiro conjunto pelos elementos correspondentes do segundo conjunto. Ele é usado para traduzir caracteres, como no exemplo:

```
#!/bin/bash
senha="shellscript"
echo "senha antes: $senha"
senha=`echo $senha | tr "a-z" "A-Z"`
echo "senha depois: $senha"
```

No qual a variável senha tem seus todas as suas letras traduzidas de minúsculas para maiúsculas (tr "a-z" "A-Z").

Exercício de fixação 2 
Metacaracteres

Acesse o arquivo */etc/passwd* utilizando o comando *egrep*. Crie comandos que selecionem apenas o que está sendo pedido:

- Apenas as linhas do usuário *root*.
- Linhas que começam com vogal.
- Linhas que tenham "root" ou "www" no início.
- Logins de 4 caracteres.
- Linhas que não terminam com ":".
- Números com no mínimo de 3 dígitos.

Roteiro de Atividades 7

Atividade 7.1 – Verificando a existência de arquivos

Crie um script que recebe como parâmetro um nome de arquivo e verifica se este arquivo existe.

```
#!/bin/bash
if [ _____$1 ]


________
   echo “O arquivo $1 existe”


________
   echo “Arquivo inexistente”
fi
```

Atividade 7.2 – Verificando a entrada de parâmetros

Modifique o script da Atividade 7.1 de forma que ele teste se realmente recebeu um parâmetro.

Atividade 7.3 – Executando sequências de comandos

Estenda o script da Atividade 7.2 para escrever o tamanho do arquivo, caso ele exista.

Utilize o comando *ls -lh*, que traz a informação do arquivo, combinado com o comando *cut*, para trazer somente o campo do tamanho do arquivo.

```
#!/bin/bash
if [ $# __________ ]
then
   echo “Script que verifica se um arquivo existe”
   echo “Uso: __________ arquivo”
   __________
else
   if [ __________ ]
   then
       echo “ Tamanho do arquivo $1: `________ | ________ “ “ ________ `”
   else
        echo “Arquivo inexistente”
```

```
   fi
fi
```

Atividade 7.4 – Combinando parâmetros, leitura e execução de comandos

Crie um script que auxilie na criação de arquivos. Ele deve receber como parâmetro um nome e criar o arquivo no diretório corrente, com a permissão de escrita. Em seguida, deve perguntar ao usuário se ele quer editar o arquivo ou não. Se sim, abrir um arquivo em um editor. Verifique se o usuário passou o número correto de parâmetros (1) e se o arquivo que ele pediu para criar já existe.

```
#! /bin/bash
if [ __________ ]
then
    _________ [ _______________ ]
    _________
            echo “este arquivo já existe!”
    else
       _______________
       chmod ____________ $1
       echo “ O arquivo _______ foi criado”
       echo _______ “Deseja edita-lo agora (s/n)? “
       _______________
       if [ $opcao = ____________ ]
       _______________
            vi _________
       fi
   fi
```

Atividade 7.5 – Utilizando comando grep/egrep

Crie um script que solicita a informação de um nome de usuário e verifica se o nome informado é de um usuário válido no sistema (desconsiderando as diferenças entre maiúsculas e minúsculas).

```
#!/bin/bash
_____________________
then
   echo “Script que verifica se um usuário é válido no sistema.”
   echo “Uso: $0 nome_usuário”
```

```
   echo “Você deve passar o nome de um usuário”
   ____________________
else
   ____________________________________
   ____________________________________
   then
   ____________________________________
   else
   ____________________________________
   fi
fi
```

Atividade 7.6 – Ainda usando grep/egrep

Crie um arquivo chamado “agenda” contendo nomes e telefones, no formato: *nome sobrenome telefone*. Crie um script que recebe como parâmetro um nome e imprima as linhas do arquivo “agenda” que o contenham. Verifique se realmente foi passado um parâmetro.

```
#!/bin/bash
_____________________
then
   ____________________________________________________
   ____________________________________________________
   ____________________________________________________
   exit 1
else
   ____________________________________________________
   ____________________________________________________
  then
     echo “recuperando informações”
     grep __________ agenda
  fi
fi
```

Atividade 7.7 – Combinando comandos

Crie um script que armazena em uma variável a quantidade de arquivos do diretório corrente e em seguida escreve o valor dessa variável. Utilize os comando *pwd* para ver o diretório corrente, *ls* para lista-lo e *wc* para contar.

```
#!/bin/bash

dir =  ____________________

quant = ____________________

if [ ____________________ ]

then

  echo “O diretório corrente _____________ tem _____________ arquivos”

else

  echo “Não há arquivos no diretório corrente _____________________”

fi
```

Atividade 7.8 – Aninhando condicionais

Crie um script que recebe três números como parâmetros e exiba o maior deles. Verifique se realmente foram passados três parâmetros.

```
#!/bin/bash

_____________________________________________________

then

   echo “Script que verifica qual é o maior entre 3 números”

   echo “Entre com os números nos parâmetros”

   echo “Uso: $0 numero1 numero2 numero3”

   exit 1

else

   if ___________________ && ___________________

   then

      maior= ___________________

   elif ___________________ & ___________________

   then

      maior= ___________________

   else

      maior= ___________________

   fi
```

```
    echo "O número _____ é o maior deles"
fi
```

Atividade 7.9 – Utilizando expressões regulares

1. Crie um script que recebe duas palavras como parâmetro e verifica se a primeira está contida na segunda.
2. Crie um script que recebe o nome de um arquivo texto como parâmetro, verifica se o usuário possui permissão de leitura deste arquivo e, caso tenha, exiba as seguintes informações sobre o arquivo: número de caracteres, número de palavras e número de linhas do arquivo (utilize o comando *wc*).

Atividade 7.10 – Utilizando *case*

Crie um script que retorna o dia da semana por extenso, a partir do comando *date*. Exemplo: hoje é segunda-feira.

Atividade 7.11 – Utilizando *case* com menu de opções

Crie um script que apresenta ao usuário um menu com as seguintes opções:

1. Exibir status da utilização das partições do sistema (df -h);
2. Exibir relação de usuário logados (who);
3. Exibir data/hora (date);
4. Sair.

A partir daí, receba a opção digitada pelo usuário e execute o comando respectivo. Informe ao usuário caso ele esteja digitando uma opção fora das esperadas.

Atividade 7.12 – Combinando *if* com *case*

Crie um script que realiza as operações aritméticas básicas (soma, subtração, divisão e multiplicação), recebendo a operação e os operandos como parâmetros. Verifique se os operandos são numéricos, com sinal ou sem sinal.

Use o padrão de comparação das expressões regulares para verificar se o parâmetro contém somente números: [*"$2" =~ ^[+-]?[0-9]*$]*

Atividade 7.13 – Combinando *if*, *case* e comandos

Crie um script que recebe o nome de um arquivo como parâmetro e que, caso este arquivo não exista, escreva a lista de arquivos de */home* neste arquivo. Se o arquivo existir, exibe uma mensagem dizendo o arquivo já existe; em seguida dar a opção de apagar o arquivo anterior, substituindo seu conteúdo, ou a opção de adicionar no final do arquivo a nova informação.

8

Shell Script

objetivos

Aprender a controlar o fluxo de execução dos scripts com o uso de repetições, programar de forma modular e utilizar variáveis compostas do tipo *lista*.

conceitos

Estruturas de repetição, funções e arrays.

Estruturas de repetição

As estruturas de repetição servem para executar uma sequência de comandos várias vezes. Existem dois tipos de comandos de repetição:

- Baseados em contadores:
  - Comando *for.*
- Condicionais:
  - Comando *while.*
  - Comando *until.*

Cada repetição de comandos é chamada de iteração.

Uma estrutura de repetição é usada na programação quando se deseja que uma determinada sequência de comandos seja executada várias vezes. Existem dois tipos de comandos de repetição: os baseados em contadores (*for*) e os condicionais (*while* e *until*).

Comando *for*

- Bastante utilizado em várias linguagens de programação.
- Forma mais comum é utilizar contador com incremento/decremento e condição de parada.
- Sintaxe 1 do comando:

```
for ((expressao1; expressão2; expressão3))
do
<comandos>
done
```

Semântica do comando:

- A primeira expressão é executada na inicialização do comando, e geralmente atribui um valor inicial para uma variável.
- A segunda expressão é avaliada a cada iteração do comando, e geralmente é uma comparação de uma variável com uma condição de parada.
- A terceira expressão é executada e utilizada para incrementar ou decrementar a variável de controle do laço.

O comando *for* é bastante utilizado em várias linguagens de programação. Na sua forma mais comum, um grupo de comandos é executado com base no valor de uma variável que é inicializada ao se iniciar o comando *for*. Ao final de cada loop (iteração) do comando a variável é modificada; a partir daí, verifica-se uma condição de parada, que se alcançada faz parar o comando. Assim, o comando *for* pode ser expresso como:

```
for ((expressao1; expressão2; expressão3))
do
   <comandos>
done
```

A primeira expressão é executada na inicialização do comando, e pode servir para atribuir um valor inicial para uma variável. A segunda expressão é avaliada a cada iteração do comando, e tipicamente é utilizada a comparação de uma variável com uma condição de parada. Por fim, ao final de cada iteração, a terceira expressão é executada e utilizada para incrementar ou decrementar a variável de controle do laço. O exemplo seguinte inicializa uma variável com o valor 1, compara o valor desta variável com 10, executa o comando *echo* e, por fim, incrementa o valor da variável de controle *x* em uma unidade (x++):

```
#!/bin/bash
for (( x=1; x < 10 ; x++ ))
do
  echo $x
done
```

O comando é executado até que a variável *x* atinja o valor 10.

- Embora esta seja a forma comumente utilizada em outras linguagens de programação, no bash a forma mais comum de utilizar o comando *for* é percorrendo os valores de uma lista.
- Essa lista pode ser de valores de uma variável ou a saída de um comando.

```
for <variavel> in <lista>
do
  <comando>
done
```

No próximo exemplo, o comando *seq* é utilizado para gerar valores de 1 a 9. A variável *x* assume cada um destes valores a cada iteração do comando *for*:

```
#!/bin/bash
for x in `seq 1 9`
do
   echo $x
done
```

- O comando *seq* é utilizado para gerar valores de 1 a 9.
- A variável *x* assume cada um destes valores a cada iteração do comando *for*.
- Uma "lista" é uma string na qual os elementos estão separados por caracteres especiais.
  - No bash usa-se a variável especial IFS para definir o separador dos elementos da lista.
    - O valor padrão é um sinal de tabulação, uma nova linha ou um espaço em branco.

Para o bash, uma "lista" é basicamente uma string na qual os elementos estão separados por caracteres especiais. O bash utiliza o valor de uma variável especial, chamada IFS, como separador de elementos. O valor padrão desta variável considera um sinal de tabulação, uma nova linha ou um espaço em branco como separadores de elementos. O exemplo abaixo demonstra isso:

```
#!/bin/bash
valores="a b c 1 2 3"
for i in $valores
do
  echo $i
done
```

- O comando *for* percorre toda a lista escrevendo os valores um a um.
- É possível modificar o valor da variável IFS para que o bash utilize outro caractere como separador de elementos de uma lista, tal como no exemplo abaixo:

```
#!/bin/bash
IFS=";"
lista="a1;a2;a3;a4"
for i in $lista
do
   echo "elemento: $i"
done
```

Exercício de fixação 1
Alterando o valor da variável IFS

Modifique o valor da variável IFS para que o bash utilize outro caractere como separador de elementos de uma lista.

Comandos *while* e *until*

Os comandos *while* e *until* são utilizados para executar um bloco de comandos uma ou mais vezes mediante uma condição de parada. Eles diferem entre si apenas na forma de interpretação.

Comando *while*:

- O comando *while* realiza iterações enquanto a expressão da condição for avaliada como verdadeira

Comando *until*:

- O comando *until* realiza iterações até que a expressão da condição seja avaliada como verdadeira.

Os comandos *while* e *until* são utilizados para executar um bloco de comandos enquanto uma determinada expressão for avaliada como verdadeira.

O comando *while* realiza iterações enquanto a expressão for avaliada como verdadeira; já o comando *until* funciona até que a condição seja avaliada como verdadeira.

Sintaxe do comando *while*:

```
while <condição>
do
  <comandos>
done
```

Sintaxe do comando *until*:

```
until <expressao>
do
  <comandos>
done
```

O exemplo seguinte escreve os valores de 1 a 9 utilizando o comando *while*:

```
#!/bin/bash
x=1
while ((x < 10 ))
```

```
do
   echo $x
   x=$((x+1))
done
```

Lembre-se de que utilizamos os duplos parênteses para avaliar expressões aritméticas.

O mesmo código utilizando *until* seria:

```
#!/bin/bash
x=1
until ((x >= 10 ))
do
   echo $x
   x=$((x+1))
done
```

Funções

- São usadas na programação para promover a organização do código em "blocos" de funcionalidades.
- Também são chamadas de módulos, subprogramas ou sub-rotinas.
- Permitem a reutilização dos códigos.
- Capazes de aceitar valor e retornar resultados.
- São implementadas dentro dos scripts e definidas por padrão no início deles.
- O conjunto de comandos fica encapsulado dentro da função.
- Para executar os comandos basta invocar o nome da função ao longo do script.
  - Quando isso acontece, o fluxo de execução, que estava seguindo linha a linha, de cima para baixo, é deslocado para o conteúdo da função.

As funções são usadas na programação para promover uma maior organização do código em "blocos" de funcionalidades (chamados de módulos), que o encapsula de forma autocontida. Assim, permite-se a reutilização dos códigos em outros cenários.

Cada função é considerada como um subprograma, uma sub-rotina ou um módulo, capaz de aceitar valor e retornar resultados. As funções são implementadas dentro dos scripts e são por padrão definidas no início deles.

A sintaxe de uma função é:

```
function nome_da_função ( ){
    <comandos>
}
```

Onde: *Nome_da_função* é um nome que o programador escolhe para representar aquela sequência de código. Ele deve seguir a mesma lei de formação de variáveis, começando por letra e seguido por letras e/ou dígitos, com o caractere "_" permitido para ligar palavras. Assim, o conjunto de comandos é encapsulado e só é executado quando o nome da função é invocado ao longo do script. Quando isso acontece, o fluxo de execução que estava seguindo linha a linha, de cima para baixo, é deslocado para o conteúdo da função.

Vamos considerar o exemplo da repetição mostrado anteriormente, cuja função é escrever os números de 1 a 9. A sequência de código foi colocada dentro de uma função chamada *conta*.

```
#!/bin/bash
x=1
function conta ( )
{
until ((x >= 10 ))
do
   echo $x
   x=$((x+1))
done
}

echo "Realizando a contagem de 1 a 9"
conta
echo "Saindo do script"
```

- Função *conta* escreve os números de 1 a 9.
- O código da função deve necessariamente ser descrito antes de a função ser invocada.

No exemplo, o primeiro comando a ser executado é o comando *echo* "Realizando a contagem de 1 a 9", pois ele está no fluxo normal da execução. Após ele ser executado, passa-se então para o próximo comando, que é a invocação ou chamada da função *conta*. Neste momento, a execução é desviada para o conteúdo da função, executando-se o comando *until*. Após serem feitas as iterações necessárias dentro do comando de repetição e não havendo mais comandos dentro da função, a execução retorna para o próximo comando depois da chamada da função, que é a execução do comando *echo* "Saindo do script". Após sua execução, o script é encerrado.

O código da função deve necessariamente ser descrito antes de a função ser invocada. Por esse motivo é que se aconselha que as funções sejam especificadas no início do script.

Ideia da modularização e da reutilização de uma função:

- Que elas não sejam dependentes de outras partes do script, ou seja, não utilizem de valores de variáveis definidas fora da função.

Como conseguir isso?

- A comunicação do resto do script com a função pode ser feita através de parâmetros, da mesma forma que os parâmetros posicionais são passados aos comandos.

Passagem de parâmetros:

- Permite a generalização da função simplesmente mudando o valor passado por parâmetro.

Para garantir a modularização e a reutilização de uma função em outro script qualquer, os dados de que ela necessita não podem ser dependentes de outras partes do script. No exemplo anterior, a contagem começa no valor atribuído à variável *x*, que é definida fora da função. Assim, se usarmos a função *conta* em outro script, precisaríamos da garantia de que haveria uma variável *x* também definida com valor 1 no outro script. Isso nem sempre é possível ou desejável. A ideia é que o valor seja comunicado à função independente de existir a variável *x*. Essa comunicação é feita através de parâmetros. Da mesma forma que passamos parâmetros aos comandos, passamos também às funções, colocando os valores após o nome da função, quando vamos chamá-la. Daí, a função recebe os valores através dos parâmetros posicionais.

Por exemplo:

```
#!/bin/bash


function conta ( ){
x=$1
until ((x >= 10 ))
do
   echo $x
   x=$((x+1))
done
}


echo “Realizando a contagem de 1 a 9”
conta 1
echo “Saindo do script”
```

- O valor 1 é passado junto com o nome da função conta, por parâmetro.
- Dentro dela a variável *x* recebe esse valor.

No exemplo, temos que o valor 1 é passado junto com o nome da função *conta*. Só dentro dela que a variável *x* recebe esse valor. Isso garante que essa sequência de código possa ser usada em qualquer outro script, sem a necessidade de haver uma variável *x* definida nele.

Esta técnica permite ainda que esse valor seja flexibilizado, ou seja, se for passado o valor 2 junto ao nome da função, a contagem passa a ser feita a partir de 2 e não mais de 1.

Os parâmetros são percorridos também como uma lista pelo comando de repetição *for*. Por exemplo:

```
#!/bin/bash
function imprime_parametros ( ){
  echo “Esses são os parâmetros recebidos pela função”
  for i in $*
  do
      echo $i
  done
}
imprime_parametros um dois tres quatro
```

O comando *for* percorre a lista de parâmetros ($*) e o comando *echo $i* escreve cada um deles.

Da mesma forma que função recebe valores através dos parâmetros, ela é capaz de retornar valor. Na verdade ela retorna o status de execução. Esse valor é retornado através do comando *return*, como por exemplo:

```
#!/bin/bash
function impar ( ){
   res=$(($1%2))
   if [ $res -ne 0 ]
   then
          return 0
   else
          return 1
   fi
}
impar 3
echo $?
```

- Para o valor 3 passado por parâmetro, a função retorna 0.
- Esse valor é retornado na variável $?

Assim, ao chamar a função *impar* com o parâmetro 3, é verificado que o resto da divisão de 3 por 2 (*res=$(($1%2))*) é diferente de zero, e portanto, ímpar. Assim, executa-se o comando *if*, retornando valor 0. Esse valor é retornado na variável $?.

Podemos usar esse conceito para retornar direto na condição do comando *if*.
Como por exemplo:

```
#!/bin/bash
function impar ( ){
   res=$(($1%2))
   if [ $res -ne 0 ]
   then
       return 0  #sucesso
   else
       return 1  #falha
   fi
 }
if impar 3
then
  echo “número ímpar”
else
   echo “número par”
fi
```

Com o mesmo valor 3 passado por parâmetro, e o retorno 0 da função. A condição estando com valor 0 implica sucesso do comando. Portanto, aciona o comando *echo* que escreve “número ímpar”.

Lembrando que para o bash, o valor 0 significa sucesso de execução de um determinado comando. Ao colocar a função *impar* como condição do *if* e ela retornando 0, como no caso do exemplo, significa para o comando que foi bem-sucedido, tornando assim a condição verdadeira; portanto, será executado o comando *echo* “número ímpar”.

No entanto, é permitido ao comando *return* retornar apenas valores numéricos. Valores diferentes de números devem ser retornados através do comando *echo*, como por exemplo:

```
#!/bin/bash
function teste()
{
    echo “aqui retorna-se o valor não numérico da função”
    return 23
}
valor=$(teste)
echo “ o status da função retornou $?”
echo “retorno == $valor”
```

A execução do comando *valor=$(teste)* faz com a função teste seja chamada e o retorno dela seja atribuído à variável *valor.* Este retorno é ditado pelo conteúdo expresso no comando *echo "*Aqui retorna-se o valor não numérico da função". Ao invés de escrever na tela, ele "escreve" na variável.

A execução do comando *valor=$(teste)* faz com que a função teste seja chamada, e o retorno dela seja atribuído à variável *valor.* No entanto, este retorno é ditado pelo conteúdo expresso no comando *echo "aqui retorna-se o valor não numérico da função"* que, quando usado desta forma na função, é capaz de retornar um valor não numérico. O exemplo mostra ainda a combinação dos dois retornos, usando também o comando *return* para retornar na variável de status *S?* o valor 23.

Para aumentar ainda mais o conceito de encapsulamento, fazendo com que as funções se comportem como módulos independentes e possam ser portadas/reutilizadas em qualquer outro script, é desejável que qualquer variável usada dentro da função seja também específica dela. Esse efeito é conseguido através do conceito de criação de variáveis locais. Elas têm esse nome porque são vistas somente dentro do escopo da função a qual foram declaradas. Por exemplo:

```
#!/bin/bash
x=20
function conta ( )
{
local x=1
until ((x >= 10 ))
do
   echo $x
   x=$((x+1))
done
echo "imprimindo x dentro da função: -$x-"
}
echo "Realizando a contagem de 1 a 9"
conta
echo "imprimindo x fora da função: -$x-"
echo "Saindo do script"
```

O valor de *x* local só é visto/modificado dentro da função.

Aqui existem duas instâncias da variável *x*. Uma criada fora da função, com valor 20 e outra declarada como *local*, com valor inicial 1. Quando a variável é criada como local, passa a ser visível somente dentro do escopo da função onde foi declarada. Assim, a variável *x* que está fora da função não sofre mudanças. O comando *echo "imprimindo x dentro da função: -$x-"* mostrará o valor 10 para a variável x, valor que ela terá após a iteração *until* de 1 a 9. Já fora da função, o comando *echo "imprimindo x fora da função: -$x-"* mostrará o valor 20 para a variável *x*, valor que não foi alterado pela função *conta*.

Exercício de fixação 2
Etapas do script

```
#!/bin/bash

x=20

function conta ( )

{

local x=1

until ((x >= 10 ))

do

   echo $x

   x=$((x+1))

done

echo “imprimindo x dentro da função: -$x-”

}

echo “Realizando a contagem de 1 a 9”

conta

echo “imprimindo x fora da função: -$x-”

echo “Saindo do script”
```

Arrays

- Um array é uma variável que contém uma lista de elementos.
- Esses elementos são tratados de maneira individual através de índices.
- Os índices são valores numéricos que começam em 0 e vão até o número de elementos do array menos 1.
- São usados para reunir informações de natureza comum.
  - Exemplo: uma data no formato dia, mês e ano.

Um array é uma variável que contém uma lista de elementos. Esses elementos são tratados de maneira individual através de índices. Os índices são valores numéricos que começam em 0 e vão até o número de elementos do array menos 1. São usados para reunir informações de natureza comum. Por exemplo, para armazenar uma data no formato dia, mês e ano, temos três informações que se relacionam. É desejável que estejam armazenadas juntas para facilitar o trabalho do programador.

Um array pode ser declarado através do comando:

```
declare -a nome_array
```

Os valores dentro do array são atribuídos de duas formas:

- Através de seus índices individuais:

```
nome_array[indice]=valor
```

- Ou da especificação completa da lista.

```
nome_array=(valor1 valor2 valor3.. valor N)
```

Por exemplo, o array *data* descrito acima poderia ter seus valores atribuídos desta forma:

```
#!/bin/bash

declare -a data

data[0]=10

data[1]=2

data[2]=2012
```

Colocando-se o dia no campo de índice 0 do array e mês e ano nos campos de índice 1 e 2, respectivamente. Ou então, desta forma:

```
#!/bin/bash

declare -a data

data=(10 2 2012)
```

Preenchendo-se todos os valores ao mesmo tempo.

O acesso a cada valor também é feito através dos índices. Por exemplo, para mostrar a data armazenada, podemos usar o comando *echo* desta forma:

```
#!/bin/bash

declare -a data


data[0]=10

data[1]=2

data[2]=2012


echo ${data[0]}

echo ${data[1]}

echo ${data[2]}
```

Ou então através do comando *for*:

```
#!/bin/bash

declare -a data


data[0]=10

data[1]=2

data[2]=2012

```

```
for num in 0 1 2

do

  echo ${data[num]}

done
```

As chaves são usadas no comando *echo ${data[num]}* para garantir que aquele índice específico seja considerado.

O número de elementos de um array pode ser obtido através do comando:

```
${#nome_array[*]}
```

Assim, para garantirmos que todos os elementos do array sejam alcançados no comando *for*, por exemplo, podemos utilizar este comando para definir o limite.

```
#!/bin/bash

declare -a data

data[0]=10

data[1]=2

data[2]=2012


tamanho=$((${#data[*]}-1));


for num in `seq 0 $tamanho`

do

  echo ${data[num]}

done
```

O comando *for* lista todos os elementos do array:

- *tamanho* = tamanho do vetor data (3) - 1;
- *seq* varia de 0 até 2.

A variável *tamanho* recebe o tamanho do vetor *data*, que é três menos 1, para que sirva de contagem no comando *seq*, de 0 até 2, obedecendo aos índices do array. Assim, serão recuperados cada um dos elementos e escritos através do comando *echo ${data[num]}*.

Podemos também utilizar o comando *read* para armazenar valores no array:

```
#!/bin/bash

declare -a lista


read -p “Quantos elementos serão inseridos? “ total

```

```
for ((i=0; i < $total; i++))
do
  read -p “Entre com valor: “ lista[$i]
done


total=$(($total-1))
echo “Estes foram os valores lidos:”
for num in `seq 0 $total`
do
  echo ${lista[num]}
done
```

- No primeiro comando *for* são lidos os elementos e armazenados no array *lista*.
- No segundo comando *for* são escritos todos eles.

O primeiro comando *for* acessa cada um dos índices do array *lista* com a variável *i* recebendo valores de 0 até a quantidade de elementos *total*, armazenando os valores lidos através do comando *read*. O segundo comando *for* apenas escreve todos eles.

Roteiro de Atividades 8

Atividade 8.1 – Verificando permissão dos arquivos

Crie um script que exibe para o diretório atual a lista de arquivos com permissão de execução.

Atividade 8.2 – Verificando usuários e informações

Crie um script que apresenta na tela todos os usuários cadastrados no sistema e seus respectivos diretórios "home".

Atividade 8.3 – Percorrendo um diretório com o comando *for* e utilizando contadores

Crie um script que recebe como parâmetro o caminho de um diretório e lista seus arquivos e diretórios um a um. Se for diretório, deve escrever "dir" ao lado do diretório. Numere cada uma das linhas. Lembre-se de testar se realmente foi passado um parâmetro.

Atividade 8.4 – Ainda manipulando listas com o comando *for* e contadores

Crie um script que conta o número de palavras no conteúdo do diretório passado por parâmetro. Use repetição para fazer a contagem. Lembre-se de testar se realmente foi passado um parâmetro.

Atividade 8.5 – Manipulando sequências com o comando *for*

Crie um script que recebe um número como parâmetro e exibe todos os números de 1 até o número recebido. Lembre-se de testar se realmente foi passado um parâmetro.

Atividade 8.6 – Testando os diversos comandos de repetição

Crie um script que recebe um número como parâmetro e exibe todos os números pares de 1 até o número recebido. Lembre-se de testar se realmente foi passado um parâmetro. Faça usando a repetição *for* e em seguida usando *while* ou *until*.

Atividade 8.7 – Utilizando o comando *for* como contador

Crie um script que recebe dois números passados por parâmetro e escreve o primeiro número a quantidade de vezes especificada no segundo parâmetro. Lembre-se de testar se realmente foram passados dois parâmetros.

Atividade 8.8 – Utilizando repetição condicionada

Crie um script que lê valores do teclado enquanto o valor digitado for diferente de 0.

Atividade 8.9 – Criando uma agenda simples

Crie um script que constrói uma agenda telefônica. Você deve apresentar um menu ao usuário contendo essas opções: 1 – inserir; 2 – listar; 3 – sair. O script deve aceitar opções até que seja digitada a opção de sair. Na opção 1, deve ler um nome e um telefone e armazenar nas próximas duas posições vazias de um array chamado "agenda". Assim, se "agenda" estiver vazio, o nome será armazenado na posição 0 e o telefone na posição 1. Na opção 2, o script deve listar todos os nomes com seus respectivos telefones registrados na agenda.

Atividade 8.10 – Criando uma calculadora simples utilizando funções

Crie um script que realiza as operações aritméticas básicas. Ele deve apresentar o menu (1–soma; 2 – subtração; 3 – divisão; 4 – multiplicação; 5 – sair) e fazer as operações enquanto não for digitada a opção 5. Use funções para cada uma das operações. Verifique se os operandos são numéricos, com sinal ou sem sinal.

Use o padrão de comparação das expressões regulares para verificar se o parâmetro contém somente números: [“$2” =~ ^[+-]?[0-9]*$].

9

Instalação de aplicações

objetivos

Instalar aplicações a partir de seus códigos-fontes e de arquivos binários.

conceitos

Aplicações no sistema operacional Linux.

Aplicações no sistema operacional Linux

Aplicações são instaladas em um sistema por diversos motivos, como: necessidade dos usuários e mudanças nas funções desempenhadas pelo sistema, entre outros. Se uma aplicação não estiver mais sendo utilizada, deve ser logo removida, pois pode se tornar uma porta de entrada para invasores. Também é importante que as aplicações instaladas sejam sempre atualizadas, seja para corrigir defeitos, incluir novas funcionalidades ou mesmo para corrigir vulnerabilidades que possam afetar a segurança do sistema.

Saber como instalar, remover e atualizar uma aplicação corretamente é uma habilidade fundamental para qualquer administrador de sistemas. Neste capítulo, veremos como executar essas tarefas. Mas antes disso, começaremos a entender como as aplicações são desenvolvidas, quais são suas principais características e como são distribuídas.

Antes que uma aplicação esteja pronta para uso, alguém deve projetá-la e escrevê-la. As aplicações são escritas utilizando uma ou mais linguagens de programação, que nada mais são do que linguagens intermediárias entre o que os humanos entendem (linguagem natural) e o que o computador entende (linguagem de máquina).

Algumas linguagens de programação se aproximam mais da linguagem natural, pois se servem de conceitos mais abstratos e, em geral são capazes de efetuar mais tarefas com uma quantidade menor de comandos. Essas linguagens são conhecidas como “linguagens de alto nível”. Outras linguagens, conhecidas como “linguagens de baixo nível”, se aproximam mais da linguagem utilizada pela máquina e, em geral, apresentam melhor desempenho ao custo de um esforço maior de programação, já que normalmente são necessários muito mais comandos para efetuar uma mesma tarefa.

Chamamos os arquivos que contêm a aplicação escrita com uma linguagem de programação de código-fonte. Seja qual for o nível da linguagem, o código-fonte deve passar por um processo de tradução para que seja entendido pelo computador. Esse processo pode ser feito uma única vez e o resultado pode ser armazenado em um ou mais arquivos contendo o código já traduzido para a linguagem de máquina ou, ainda, pode ser feito toda vez que a

aplicação for executada. No primeiro caso, o processo de transformar o código-fonte em um arquivo contendo código de máquina (arquivo binário) é chamado de *compilação*. Quando esse processo de tradução acontece, sempre que a aplicação é executada, é chamado de *interpretação*. No capítulo anterior, quando foi vista a programação em shell script, vimos que os programas criados são interpretados todas as vezes em que são executados por um interpretador, que neste caso é o próprio shell.

Linguagens de programação

- Alto nível – C++, Java.
- Baixo nível – Assembly.
- Linguages compiladas – C, C++, Pascal.
- Linguagens interpretadas – Bash, Perl, PHP.

No Linux, as linguagens mais utilizadas no desenvolvimento de programas são C e C++. A linguagem C é considerada por alguns uma linguagem de "médio nível", pois apesar de possuir características de linguagens de alto nível, também possui características de linguagens de baixo nível, como o acesso direto ao hardware. A linguagem C foi criada no início da década de 1970 e desde então tornou-se a linguagem preferida para o desenvolvimento de sistemas operacionais. O kernel do Linux é escrito em C, assim como vários de seus aplicativos e bibliotecas de funções. A linguagem C++ é uma linguagem criada a partir da linguagem C, acrescentando a ela características de linguagens de alto nível, como a orientação a objetos. Um programa escrito em C precisa ser compilado antes de ser utilizado. Dependendo do seu tamanho, o código de um programa em C pode estar contido em diversos arquivos, cada um contendo um módulo ou parte do programa. A compilação de um programa é feita com o auxílio de programas como compiladores e ligadores (do inglês *linkers*). O processo de compilação pode ser simplificado através do uso do programa *make*, que lê um arquivo chamado Makefile, que contém as instruções de como compilar de maneira correta os arquivos-fontes de uma aplicação. No Linux, o compilador C mais utilizado é o GNU C Compiler, também conhecido como *gcc*.

Uma das características dos sistemas Unix é a modularidade. Seus comandos, em geral, desempenham tarefas simples e as tarefas mais complexas são executadas utilizando dois ou mais comandos. De forma análoga, programas complexos são criados através da combinação de funções contidas em módulos menores, também chamados de bibliotecas. Para entender melhor este conceito, vamos utilizar um exemplo: imagine um programador que deseja escrever um programa capaz de tocar arquivos mp3. O programa deverá ser capaz de abrir um arquivo mp3, decodificar o seu conteúdo e reproduzi-lo utilizando a placa de som, além de prover uma interface para o usuário que permita executar as ações de parar, iniciar, avançar e retroceder a posição da música. Ele poderá escrever um código capaz de fazer todas essas funções, ou poderá utilizar bibliotecas de funções criadas por outros programadores que disponibilizem as funções descritas acima, facilitando o desenvolvimento do programa.

As funções de uma biblioteca podem ser utilizadas por um programa de duas maneiras: estaticamente ou dinamicamente. No modo estático, o código da função utilizada pelo programa é embutido no binário do programa. Desta forma, o programa pode ser executado em um sistema sem que haja a necessidade deste possuir as bibliotecas com as funções que o programa utiliza. Esta forma apresenta, como desvantagem, o tamanho dos arquivos binários dos programas, que passam a ser bem maiores. Por outro lado, um programa pode utilizar uma biblioteca de função de maneira dinâmica. Neste caso, quando o programa é executado, ele procura no sistema as bibliotecas de que necessita e as carrega na memória.

A vantagem de utilizar esta forma é que vários programas podem utilizar a mesma biblioteca já carregada na memória, diminuindo assim o total de memória utilizada pelo sistema como um todo. Além disso, programas que utilizam bibliotecas carregadas dinamicamente possuem arquivos binários menores.

No Linux, os arquivos contendo bibliotecas dinâmicas se localizam geralmente nos diretórios */lib* e */usr/lib*. Geralmente estes arquivos são nomeados utilizando o padrão *libXXX.so[.z.y]*, onde *XXX* é o nome ou sigla da biblioteca, *.so* é a extensão que identifica bibliotecas dinâmicas e *.z.y* é a versão da biblioteca, que pode ser opcional. Dado um arquivo binário qualquer, podemos identificar as bibliotecas que ele utiliza com o comando *ldd*. O exemplo a seguir mostra a lista das bibliotecas que o shell *bash* utiliza:

```
# ldd /bin/bash

Linux-gate.so.1 => (0xffffe000)

libncurses.so.5 => /lib/libncurses.so.5 (0xb7f82000)

libdl.so.2 => /lib/tls/libdl.so.2 (0xb7f7e000)

libc.so.6 => /lib/tls/libc.so.6 (0xb7e46000)

/lib/ld-Linux.so.2 (0xb7fd1000)
```

A saída do comando apresenta o nome das bibliotecas utilizadas, o caminho completo do arquivo onde estão contidas e um número que identifica a posição de memória que ocupam.

Quando um programa necessita carregar uma biblioteca, ele utiliza um carregador em tempo de execução (do inglês *run-time linker*), conhecido como *ld.so*. Esse carregador, por sua vez, precisa identificar os diretórios que contêm as bibliotecas. Para isso, ele utiliza o arquivo */etc/ld.so.cache,* que é gerado através da execução do comando *ldconfig.* Este comando, quando executado, lê o arquivo */etc/ld.so.conf*, que contém os diretórios do sistema onde os arquivos de bibliotecas podem ser encontrados. Sempre que o arquivo */etc/ld.so.conf* for alterado, o comando *ldconfig* deve ser executado para atualizar o arquivo */etc/ld.so.cache*.

Quando um programador decide distribuir sua aplicação, ele pode fazê-lo de duas maneiras: distribuir os arquivos-fontes ou a aplicação já compilada. Veremos, a seguir, como proceder para instalar aplicações distribuídas nas duas formas citadas. Vale lembrar que uma aplicação não é composta apenas pelo seu arquivo binário. Em geral, uma aplicação possui também arquivos de configuração, manuais de sistema e arquivos de documentação, além de exemplos. Chamamos esse conjunto de arquivos de pacote.

Reaproveitamento de código:

- Uso de bibliotecas estáticas ou dinâmicas.
  - Vantagens: tamanho dos programas fica reduzido e as bibliotecas podem ser compartilhadas com outros programas.
  - Desvantagens: a compilação e execução do programa só é possível se a biblioteca estiver instalada no sistema.
- Em geral, arquivos de bibliotecas possuem o formato: *libXXX.so[.z.y]*.
- O comando *ldd* identifica as bibliotecas usadas por uma aplicação.

Instalando aplicações a partir de seus códigos-fontes

O método de instalação a partir de arquivos-fontes é o método que envolve mais trabalho para quem vai instalar; porém, envolve menos trabalho para o programador, já que ele deve apenas disponibilizar os arquivos-fontes. A grande maioria das aplicações escritas para sistemas Unix tem seu código-fonte disponibilizado, possibilitando a todos o acesso ao modo como o programa foi escrito. É comum que programas licenciados pela General Public License (GPL) ou por uma licença open source estejam disponíveis nessa forma e que programas proprietários não estejam.

Embora esta seja a forma de instalação mais trabalhosa, este método permite que o programa seja modificado e personalizado. Em geral, é possível escolher onde o programa deverá ser instalado e onde procurará por arquivos de configuração. No entanto, o grau de personalização de um programa depende muito de sua função e do seu programador. Imagine um programa que funcione como um cadastro de telefones. Este programa poderia armazenar suas informações em um dos diversos Sistemas Gerenciadores de Bancos de Dados (SGBDs) disponíveis para Linux, ou poderia simplesmente armazenar as informações em arquivos textos. O programador que desenvolveu esse aplicativo poderia permitir que o administrador escolhesse o modo de armazenamento dos dados gerados pelo programa por meio de opções especiais utilizadas no processo de compilação. Ativando ou desativando essas opções, os requisitos para a instalação do programa e o seu tamanho se alteram já que, por exemplo, ao desativar o suporte para armazenagem de dados em bancos de dados, tal programa não iria requerer bibliotecas com funções de acesso a bancos de dados. Usualmente, o processo de instalação de um programa a partir de um arquivo-fonte é composto de quatro passos, que serão descritos a seguir.

Obtenção dos arquivos-fontes

A obtenção dos arquivos-fontes pode se dar através de um DVD ou, mais comumente, através de sites como o Sourceforge, que funciona como um imenso repositório de projetos de software livre e disponibiliza milhares de programas. Os códigos-fontes geralmente são distribuídos em arquivos compactados no formato "tar.gz" ou "tar.bz2". Os exemplos seguintes mostram como descompactar arquivos nos formatos "tar.gz" e "tar.bz2", respectivamente.

Visite o site do desenvolvedor Sourceforge: http://www.sourceforge.net.

```
# tar xvfz programa.tar.gz

# tar xvfj programa.tar.bz2
```

Verificação do ambiente para a compilação

Configurando a aplicação:

- Buscar documentação: arquivos INSTALL e README.
- Script de configuração: *./configure*.

Problema de dependência:

- Verificar se todas as bibliotecas necessárias estão instaladas.

Após descompactar o arquivo, é recomendável ler o conteúdo de arquivos como INSTALL e/ou README, caso estejam presentes, pois contêm informações sobre como compilar e instalar o programa em questão. A verificação do ambiente, necessária para a compilação, é feita através de um script auxiliar chamado *configure*, presente na grande maioria dos programas. Esse script verifica se todos os pré-requisitos necessários para a compilação do programa estão presentes e gera o arquivo *Makefile*. O script *configure* geralmente possui a

opção "—help", que lista todas as opções de configuração de um programa. O comando a seguir executa o script *configure*:

```
# ./configure
```

Como vimos, os programadores escrevem programas utilizando funções presentes em bibliotecas de funções. Mas só as bibliotecas não bastam na hora de compilar um programa, sendo necessário também que estejam presentes seus arquivos de cabeçalho (*headers*). Nas linguagens *C* e *C++*, antes de uma função ser utilizada, precisa ser definida, ou seja, é necessária uma espécie de descrição contendo o nome da função, seu tipo e os parâmetros que aceita. Para que não seja necessário manter no sistema vários arquivos com todos os códigos de funções, os projetistas dessas linguagens pensaram num jeito de separar do seu código as declarações das funções. Assim, ao compilar um programa, é necessário apenas ter essas declarações, que ficam contidas em arquivos com a extensão *.h.* Para satisfazer os pré-requisitos para a compilação de um programa (os arquivos de cabeçalho e as bibliotecas que o programa utiliza), ainda será preciso instalar e compilar as bibliotecas necessárias, ou instalá-las por intermédio de pacotes da distribuição, como veremos adiante.

Compilação

Compilando e instalando a aplicação com *Makefile*:

- make
- make install

Uma vez configurados e satisfeitos os pré-requisitos para a compilação do programa, podemos iniciar o processo de compilação utilizando o comando *make*, que na verdade não é um compilador, mas apenas lê arquivos especiais (*Makefiles*), que contêm regras para compilar o programa. Essas regras especificam os arquivos-fontes que devem ser compilados, o compilador que deve ser utilizado, o modo como esses arquivos-fontes devem ser ligados e os diretórios onde devem ser armazenados ao serem instalados, entre outros aspectos. Sua utilização é bastante simples, bastando executá-lo no diretório em que se encontra o arquivo *Makefile*, como mostra o comando a seguir.

```
# make
```

Instalação

O processo de compilação tanto pode ser rápido, para programas simples, quanto pode durar horas, dependendo do tamanho e da complexidade do código-fonte e da velocidade da máquina. Por fim, a instalação propriamente dita é feita pelo programa *make*, usualmente utilizando-se a opção *install*, que em geral especifica uma regra especial dentro do *Makefile* que faz com que o *make* copie os arquivos do programa já compilado para os diretórios onde deverão ser instalados no sistema. Para fazer a instalação, basta executar o comando abaixo:

```
# make install
```

Isso fará com que o *make* copie os arquivos contendo o código compilado, manuais e arquivos de configuração para os diretórios adequados. Note que, embora os outros dois passos possam ser executados com uma conta de usuário comum, este último passo deve ser feito utilizando a conta *root*.

Por convenção, os programas compilados pelo administrador são instalados nos diretórios apropriados (bin, sbin, etc, var) dentro do diretório /usr/local. Isso é feito para que haja uma separação dos programas instalados dessa forma dos programas instalados por meio de

pacotes da própria distribuição. Vale observar que, ao contrário destes últimos, os programas instalados através de código-fonte não são registrados em uma base de dados e não podem ser removidos e atualizados de forma automática. Alguns programas podem ser removidos por intermédio do comando make uninstall. A opção uninstall usualmente especifica uma regra dentro do Makefile que contém os comandos para remover os arquivos instalados. Embora essa seja uma prática comum, não são todos os arquivos-fontes que apresentam essa opção. Uma boa prática, ao se instalar programas dessa forma, é utilizar um programa auxiliar como o checkinstall, que registre os arquivos que foram inseridos no sistema, facilitando, assim, a sua remoção e atualização no futuro. A instalação de programas utilizando o código-fonte é recomendada para os casos em que não exista um pacote binário pronto na distribuição, ou em casos onde é necessário alterar o código-fonte da aplicação. Para os outros casos, prefira utilizar os pacotes disponibilizados pela distribuição.

Recomendações:

- Deixar os arquivos separados e organizados em */usr/local*.
- Lembrar dos arquivos e lugares onde foram instalados. Dica: *checkinstall*.

Para remover: *make uninstall* (caso exista esta opção no *Makefile*).

Instalando aplicações a partir de arquivos binários

Aplicações que utilizam arquivos binários:

- Cuidados adicionais: dependências, local dos arquivos.
- Pacotes binários de distribuições:
  - Debian/Ubuntu – pacotes no formato *deb*.
  - Red Hat/Fedora/CentOS – pacotes no formato *rpm*.

Nessa forma de instalação, as aplicações são distribuídas em arquivos que contêm a aplicação na sua forma já compilada (por isso o nome "arquivos binários"), necessitando apenas de que sejam armazenadas nos diretórios adequados. Faremos aqui uma pequena distinção entre dois tipos de arquivos binários: o primeiro tipo é criado pelo desenvolvedor do programa, e o segundo é um tipo especial de arquivo binário criado exclusivamente para uma determinada distribuição Linux. Veremos o primeiro tipo agora e o segundo mais adiante.

A instalação de uma aplicação a partir de arquivos binários é bem mais simples do que a instalação a partir de arquivos-fontes. Geralmente, é necessário apenas obter o arquivo contendo a aplicação, descompactá-lo e copiar os arquivos para os diretórios corretos. Esse último passo pode ser feito por um script distribuído juntamente com a aplicação, especificamente para esse fim. Neste tipo de instalação, não há como o desenvolvedor garantir que o seu sistema contenha todos os pré-requisitos para que a aplicação funcione adequadamente. Sendo assim, é comum que ele adote uma das duas estratégias: desenvolver a aplicação de modo que ela requeira um conjunto mínimo de bibliotecas, deixando por conta do usuário a instalação delas, ou incluir as bibliotecas estaticamente na aplicação, deixando ela pronta para uso, não necessitando de nenhuma biblioteca adicional. A distribuição de aplicações em arquivos binários é usualmente utilizada por empresas que produzem programas proprietários e que têm a necessidade de manter seus códigos-fontes inacessíveis para terceiros (a partir de um arquivo binário, é praticamente impossível obter o código-fonte original).

Para facilitar a disseminação, instalação e sobretudo a manutenção de um sistema, as distribuições Linux criaram o que chamamos de pacotes. Um pacote é um arquivo que

contém uma aplicação ou biblioteca, scripts de instalação e dados como sua descrição e pré-requisitos. Os pacotes são manipulados por meio de programas especiais que, além de instalar, desinstalar ou atualizar um pacote, gravam informações sobre eles em uma base de dados no sistema. No Linux, os pacotes nos formatos *rpm* e *deb* são os dois tipos de pacotes mais populares. O padrão de pacotes *rpm* foi originalmente criado pela Red Hat e, posteriormente, adotado por outras distribuições como SUSE e Fedora. Os pacotes *deb* foram criados pela distribuição Debian, e são utilizados também pelas distribuições derivadas e baseadas nela, como a distribuição Ubuntu.

O funcionamento de ambos os tipos de pacotes é bastante semelhante, mudando apenas as ferramentas utilizadas para gerenciá-los. Veremos a seguir como trabalhar com pacotes *rpm.*

Os pacotes *rpm* são nomeados na forma:

```
<nomedopacote>- <versao-release>.<arch>.rpm.
```

O campo <*arch*> indica para qual arquitetura de hardware o pacote foi compilado. As principais arquiteturas são vistas a seguir:

- **i386** – pacotes para arquiteturas que utilizam processadores baseados na linha x86 da Intel;
- **PPC** – pacotes para arquiteturas que utilizam processadores PowerPC, utilizados pela Apple;
- **x86_64** – pacotes para arquiteturas que utilizam processadores de 64 bits;
- **SRC** – pacotes que não contêm binários, mas sim o código-fonte da aplicação, o que a torna portável para vários tipos de arquiteturas.

Podemos ver, a seguir, um exemplo de nome de pacote:

```
bash-3.0-31.i386.rpm


Nome do pacote: bash

Versão: 3.0

Release: 31

Arquitetura: i386
```

Pacotes *rpm*

```
<nomedopacote>-<versao-release>.<arch>.rpm
```

Exemplo:

```
bash-3.0-31.i386.rpm

Nome do pacote: bash

Versão: 3.0

Release: 31

Arquitetura: i386
```

Pacotes RPM

O comando *rpm* é a principal ferramenta utilizada para instalar, desinstalar, consultar e atualizar pacotes no formato Red Hat Package Manager (RPM). A Tabela 9.1 apresenta as principais opções do comando *rpm* e as ações que elas executam:

Opção	Ação
i	Instala um pacote.
e	Remove um pacote.
U	Atualiza um pacote, caso já esteja instalado, ou instala o pacote, caso não esteja instalado.
F	Atualiza um pacote apenas se o pacote já estiver instalado.
v	Habilita o modo "verbose". O comando mostrará informações adicionais durante a sua execução.
h	Habilita o modo *hash*. O comando mostrará uma espécie de barra de progresso durante a instalação dos pacotes.

Tabela 9.1
Opções básicas do comando ***rpm***.

Usualmente, utilizamos as opções *v* e *h* combinadas com as opções *i* ou *U* ao instalar ou atualizar um pacote. A sintaxe do comando *rpm* pode ser vista a seguir:

```
# rpm -<opção> pacote
```

Comando *rpm*

Ferramenta utilizada para instalar, desinstalar e atualizar pacotes. Não resolve problemas com dependências entre pacotes.

```
# rpm -<opção> pacote
```

A seguir são apresentados alguns exemplos de utilização do comando *rpm*.

Instalando um pacote:

```
# rpm -ivh chkconfig-1.3.20-1.i386.rpm
```

Atualizando um pacote:

```
# rpm -Uvh chkconfig-1.3.20-1.i386.rpm
```

Removendo um pacote:

```
# rpm -e finger
```

Observe que, para remover um pacote, só é necessário indicar o nome do pacote, ou nome-versão, e não o nome completo do arquivo que contém o pacote. Ao instalar ou remover um pacote, o comando *rpm* verifica se este possui dependências e caso o pacote a ser instalado necessite de algum pacote ainda não instalado ou o pacote a ser removido seja uma dependência de algum pacote ainda instalado, uma mensagem de erro será apresentada, como mostram os exemplos abaixo:

```
# rpm -ivh dvgrab-1.7-3.i386.rpm

error: Failed dependencies:

         libavc1394.so.0 is needed by dvgrab-1.7-3.i386

         libdv.so.4 is needed by dvgrab-1.7-3.i386

         libraw1394.so.8 is needed by dvgrab-1.7-3.i386

         librom1394.so.0 is needed by dvgrab-1.7-3.i386
```

```
# rpm -ev bzip2

error: Failed dependencies:

         bzip2 is needed by (installed) man-1.5p-4.i386

         bzip2 is needed by (installed) system-configprinter-
0.6.131-1.i386
```

Repare que o comando *rpm* apenas indica a existência de uma dependência, que deve ser instalada manualmente, caso necessário. Veremos mais adiante uma ferramenta capaz de resolver automaticamente problemas com dependências.

Dependências

Como vimos anteriormente, para funcionar adequadamente um programa pode necessitar de uma biblioteca ou até mesmo de outro programa. Quando lidamos com pacotes, chamamos esses pré-requisitos de *dependências*. Assim, podemos dizer que o pacote A "depende" do pacote B ou, ainda, que o pacote B é uma dependência do pacote A. Quando instalamos um pacote, a ferramenta utilizada para instalá-lo verifica, nas informações contidas no pacote, quais são suas dependências. Em seguida, verifica se essas dependências já estão instaladas no sistema consultando a base de dados de pacotes instalados. Dependendo da ferramenta, ela solicitará que as dependências sejam instaladas, ou tentará instalar as dependências automaticamente. De forma semelhante, ao tentar remover um pacote instalado no sistema, as mesmas verificações são efetuadas e a ferramenta indicará os outros pacotes que devem ser desinstalados juntamente com o pacote desejado. Essas verificações são feitas com o intuito de deixar o sistema sempre num estado consistente, isto é, garantindo que estejam instaladas todas as dependências de todos os pacotes instalados.

Consultando informações sobre pacotes

O comando *rpm* possui opções que permitem listar ou consultar informações de pacotes já instalados no sistema, e de arquivos contendo pacotes ainda não instalados. Para fazer consultas com o comando *rpm,* utilizamos a seguinte sintaxe:

```
# rpm -q<opções> <pacote | arquivo>
```

As opções de consulta devem ser sempre utilizadas em conjunto com a opção *-q*. A Tabela 9.2 apresenta as principais opções de consulta utilizadas no comando *rpm* e suas ações:

Opção	Ação
A	Lista todos os pacotes instalados no sistema.
f <arquivo>	Lista o pacote ao qual pertence o arquivo <arquivo>.
p <arquivo de pacote>	Consulta o <arquivo de pacote>. Utilizada para consultar um arquivo contendo um pacote ainda não instalado.
i <pacote>	Mostra informações sobre o pacote: incluindo nome, versão e descrição.
l <pacote>	Lista os arquivos que pertencem a um pacote.
R <pacote>	Lista as dependências de <pacote>.

Tabela 9.2
Opções de consulta do comando ***rpm***.

A seguir, são apresentados alguns exemplos de consultas combinando as opções da Tabela 9.2. O resultado não será mostrado aqui, mas você pode executar os comandos no seu sistema e verificar suas respostas.

Listando todos os pacotes instalados no sistema:

- # rpm -qa

Descobrindo a qual pacote pertence um arquivo:

- # rpm -qf /etc/passwd

Mostrando informações sobre um pacote instalado:

- # rpm -qi bash

Mostrando informações sobre um pacote ainda não instalado (arquivo de pacote):

- # rpm -qpi dvgrab-1.7-3.i386.rpm

Listando todos os arquivos que pertencem a um pacote:

- # rpm -ql bash

Conforme vimos, embora o comando *rpm* seja a principal ferramenta para instalar, atualizar, remover e consultar pacotes, faltam-lhe opções de como resolver automaticamente problemas com dependências ou de atualizar todo o sistema. Para resolver esse tipo de problema, foram criadas ferramentas de mais alto nível, como o Yellowdog Update Modifier (YUM) da distribuição Red Hat e o Advanced Packaging Tool (APT) da distribuição Debian.

- Como atualizar todos os pacotes do sistema com RPM?
- Como resolver dependências automaticamente com RPM?
- Ferramentas criadas para resolver estes problemas:

YUM ,

- APT.

Exercício de fixação 1 
Gerenciador de pacotes RPM

1. Para instalar um software qualquer, qual seria a opção a ser utilizada junto com o gerenciador RPM?
2. Para instalar um software qualquer e mostrar detalhes da instalação, qual seria a opção a ser utilizada junto com o gerenciador RPM?
3. Para instalar um software qualquer exibindo #, à medida que os arquivos são descompactados, qual seria a opção a ser utilizada junto com o gerenciador RPM?
4. Para atualizar um software, deve-se utilizar o comando RPM com que opções?
5. Para verificar se um pacote está instalado ou não, utiliza-se qual comando?
6. Para listar todos os pacotes instalados, qual comando devo utilizar?
7. Para remover um software, utiliza-se qual comando?

Este exercício permite encontrar pacotes no repositório do YUM, assim como proceder com sua instalação.

YUM

O YUM utiliza o conceito de repositório de pacotes, e por isso não há a necessidade de que os arquivos de pacotes sejam obtidos manualmente. Um repositório é um site especialmente preparado contendo pacotes e arquivos de índice, com informações sobre esses pacotes. O comando *yum* é capaz de consultar um repositório e automaticamente e baixar e instalar pacotes, liberando o administrador de efetuar essas tarefas manualmente.
Os repositórios do YUM estão divididos em três tipos básicos:

- **Base** – nos repositórios *base* se localizam os pacotes que fazem parte da distribuição, como os distribuídos no DVD de instalação.
- **Updates** – nos repositórios *update* estão atualizações para os pacotes do repositório *base* e pacotes que não fazem parte dos repositórios *base* nem suas atualizações se encontram nos repositórios *extra*.
- **Extras** – um repositório *extra* pode conter pacotes não suportados oficialmente pela distribuição.

Configurando o YUM

YUM utiliza repositórios divididos em: *base*, *update* e *extra*.

- Arquivo */etc/yum.conf*:
  - Tentativas de acesso aos repositórios.
  - Arquivo de log.
  - Local para armazenar os pacotes baixados.
- Arquivos *.repo - /etc/yum.repos.d/<arquivo>.repo*.

O YUM é configurado por meio do arquivo */etc/yum.conf,* que define opções como: número de tentativas de acesso a um repositório, arquivo de log, diretório onde manterá os pacotes baixados, entre outras. Já a localização dos repositórios a serem consultados é definida nos arquivos que ficam armazenados no diretório */etc/yum.repos.d*. A configuração padrão de ambos é suficiente para seu funcionamento correto.

Nesse capítulo, veremos somente como adicionar novos repositórios. Cada repositório é configurado mediante um arquivo com a extensão *.repo*, usualmente disponível no website do repositório. Um único repositório pode ter vários espelhos (*mirrors*), que funcionam como cópias idênticas do repositório original, mas ficam localizados em outros servidores, permitindo que o administrador escolha aquele que está localizado mais próximo ou que possua um link de maior capacidade.

O exemplo abaixo mostra o conteúdo do arquivo *jpackage.repo* que configura o repositório *jpackage*, que contém pacotes relacionados ao Java:

```
[jpackage6-fc17]

name=JPackage 6.0 for Fedora Core 17

baseurl=http://mirrors.dotsrc.org/jpackage/6.0/fedora-17/free/

gpgcheck=1
```

Ao configurar um repositório, é preciso verificar a versão de sua distribuição, que no exemplo acima é o Fedora Core 4. Repositórios e pacotes de outras distribuições ou versões podem não funcionar corretamente. Como medida de segurança, os pacotes *rpm* podem ser

assinados digitalmente. Dessa forma, é possível verificar se um pacote realmente veio de um determinado repositório e se não foi adulterado. Os pacotes *rpm* são assinados com a chave GPG do repositório ou do desenvolvedor e podem ser verificados utilizando-se sua chave pública. Para isso, é necessário importar essa chave pública por intermédio do comando *rpm*, conforme os exemplos abaixo:

```
# rpm --import GPG-PUB-KEY.asc

# rpm --import http://www.repositorio.com/GPG-PUB-KEY.asc
```

No primeiro exemplo, a chave pública é importada por meio de um arquivo que a contém, e no segundo a chave é importada diretamente do site.

Exemplo de arquivo de repositório para pacotes relacionados ao Java:

```
[jpackage16-fc4]

name=JPackage 1.6 for Fedora Core 4

baseurl=http://mirrors.dotsrc.org/jpackage/1.6/fedora-

4/free/

gpgcheck=1
```

Cuidados:

- Verificar a versão da distribuição.
- Verificar a assinatura digital:

```
# rpm --import GPG-PUB-KEY.asc

# rpm --import http://www.repositorio.com/GPG-PUB-KEY.asc
```

Utilizando o YUM

- Instalar, desinstalar e atualizar pacotes.
- Instalar e remover grupos de pacotes.
- Busca de pacotes (com expressões regulares).
- Resolução de problemas com dependências entre pacotes.
- Atualização automática de todos os pacotes instalados no sistema.

Assim como *rpm,* o comando *yum* também pode ser utilizado para instalar, desinstalar e atualizar pacotes. Além disso, permite a instalação e remoção de grupos de pacotes, a busca de pacotes com expressões regulares, a resolução de problemas com dependências entre pacotes e a atualização automática de todos os pacotes instalados no sistema. Veremos, a seguir, como efetuar cada uma dessas operações.

Instalando e removendo pacotes

Instalando e removendo pacotes com o *yum*:

```
#  yum install tcpdump
```

Instalando pacotes utilizando arquivos locais:

```
#  yum localinstall python-numeric-23.702.i386.rpm
```

Removendo um pacote:

```
#  yum remove tcpdump
```

Para instalar um pacote, basta executar o comando *yum* com a opção *install* seguida do nome do pacote. O *yum* consulta então os repositórios em busca da versão mais recente do pacote e em seguida faz o download dele e de suas dependências, caso necessário. Após obter todos os pacotes necessários, a instalação deles é executada. O comando do exemplo seguinte instala o pacote *tcpdump* e suas dependências:

```
# yum install tcpdump
```

O *yum* também pode ser utilizado para instalar pacotes que já se encontram armazenados no computador do usuário. Nestes casos, utiliza-se a opção *localinstall*, como mostra o exemplo:

```
# yum localinstall python-numeric-23.702.i386.rpm
```

Para remover um pacote e suas dependências, basta utilizar a opção *remove*, como mostra o exemplo:

```
# yum remove tcpdump
```

Sempre que o *yum* instala, atualiza ou remove um pacote, ele apresenta um sumário do que será feito e pede a confirmação do administrador antes de executar as ações necessárias. O comando do exemplo acima apresentará o seguinte sumário:

```
Setting up Remove Process

Resolving Dependencies

--> Populating transaction set with selected packages.

Please wait.

---> Package tcpdump.i386 14:3.8.2-13.FC4 set to be erased

--> Running transaction check

Dependencies Resolved

Package Arch Version Repository Size

Removing:

tcpdump i386 14:3.8.2-13.FC4 installed 765 k

Transaction Summary

Install 0 Package(s)

Update 0 Package(s)

Remove 1 Package(s)

Total download size: 0

Is this ok [y/N]:
```

Instalando grupos de pacotes

Instalando grupos de pacotes:

```
#  yum groupinstall “MySQL Database”
```

Para listar grupos disponíveis:

```
#  yum grouplist
```

O *yum* provê uma forma simples de instalar um grupo de pacotes relacionados a um determinado serviço ou perfil do sistema. Imagine que desejamos preparar um sistema para que funcione como um servidor de banco de dados. Para tanto, precisaríamos instalar um aplicativo Sistema Gerenciador de Banco de Dados (SGBD) e, possivelmente, bibliotecas de funções relacionadas. A opção *groupinstall* do *yum* pode nos auxiliar nesta tarefa. O exemplo abaixo instalaria o SGBD Mysql, suas dependências e programas relacionados:

```
# yum groupinstall "MySQL Database"
```

As aspas são utilizadas no exemplo acima porque o nome do grupo possui caracteres de espaço. Para obter uma lista dos grupos de pacotes conhecidos pelo *yum*, utilize a opção *grouplist*, conforme no exemplo abaixo:

```
# yum grouplist
```

De maneira análoga, para remover um grupo de pacotes utilizamos a opção *groupremove.*

Atualizando o sistema

Atualizando o sistema:

```
#  yum update
```

Atualizando somente determinados pacotes:

```
#  yum update bash tcpdump
```

Listando pacotes que devem ser atualizados:

```
# yum check-update
```

O *yum* pode ser utilizado para atualizar apenas determinados pacotes ou para atualizar todos os pacotes instalados no sistema. Quando utilizado com a opção *update,* o *yum* compara as versões de todos os pacotes instalados com as versões destes mesmos pacotes disponíveis nos repositórios, efetuando em seguida a atualização de todos os pacotes que estejam desatualizados. Para atualizar todo o sistema, basta executar o comando seguinte:

```
# yum update
```

Para atualizar somente determinados pacotes, basta acrescentar os nomes dos pacotes ao comando. Assim, para atualizar somente os pacotes *bash* e *tcpdump*, devemos executar o comando abaixo:

```
# yum update bash tcpdump
```

Outra opção é apenas listar os pacotes que deverão ser atualizados, sem proceder com a atualização propriamente dita. Isso pode ser feito com a opção *check-update*, que retorna a lista dos pacotes a serem atualizados, as novas versões disponíveis e o tipo de repositório de onde será baixado cada pacote.

Fazendo buscas e obtendo informações sobre pacotes

Listando os pacotes instalados:

```
#yum list installed
```

Listando um pacote com detalhes:

```
#yum list bash
```

Listando pacotes instalados que "casem" com a expressão regular:

```
# yum list installed c*
```

Listando os pacotes desatualizados:

```
# yum list updates
```

É possível consultar a base de dados de pacotes instalados no sistema e os pacotes disponíveis em um repositório. Também é possível listar os pacotes que atendam a um determinado padrão de busca, utilizando a opção *list* com alguns parâmetros adicionais, que serão vistos a seguir.

Listando os pacotes instalados:

```
# yum list installed
```

Listando informações sobre o pacote *bash*:

```
# yum list bash
```

Listando os pacotes instalados que comecem com a letra "c":

```
# yum list installed c*
```

Listando os pacotes desatualizados:

```
# yum list updates
```

O *yum* possui ainda um poderoso mecanismo de busca, capaz de procurar por palavras-chave, nome do pacote e sumário ou descrição do pacote. Isso é bastante útil quando precisamos de um aplicativo que execute uma determinada função, mas ainda não sabemos qual pacote utilizar. Digamos que estamos a procura de um aplicativo que funcione como um servidor web. Poderíamos, então, executar o *yum* com a opção *search*, como no exemplo abaixo:

```
# yum search webserver
```

Toda vez que o comando *yum* é utilizado, ele faz uma verificação por atualizações nos arquivos de índices dos repositórios. Para evitar a demora ocasionada por esta ação, podemos dizer ao *yum* que procure nas informações que já obteve desses repositórios sem ter que consultá-los novamente. Para isso, utilizamos a opção "-C".

Para obter a descrição de um pacote, basta utilizar a opção "info". O exemplo a seguir mostra como obter informações sobre o pacote *bash*:

```
# yum info bash
```

Busca por pacotes (usando palavra-chave):

```
# yum search webserver
```

- Opção "-C" serve para não buscar no repositório.

Para obter a descrição de um pacote:

```
# yum info bash
```

Outra opção interessante é a opção *provides*, que retorna o nome do pacote que oferece um determinado serviço ou a qual pacote pertence um determinado arquivo. Quando instalamos um programa a partir dos arquivos-fontes, podemos necessitar instalar pacotes para atender a certos pré-requisitos deste programa. Por exemplo: um determinado programa precisa de que a biblioteca *libxyz* esteja presente no sistema, mas não sabemos o pacote que fornece essa biblioteca. Poderíamos utilizar então o *yum* da seguinte maneira:

```
# yum provides libxyz
```

Para buscar um pacote que oferece um determinado serviço ou para descobrir a qual pacote pertence um determinado arquivo:

```
# yum provides libxyz.so.1
```

O cache do *yum* pode excluir informações manualmente:

```
# yum clean
# yum clean <headers|packages|all>
```

APT

Advanced Packaging Tool (APT) é uma ferramenta de gerenciamento de pacotes utilizada na distribuição Debian e suas variantes, que trata de forma automática problemas com dependências entre pacotes. O APT possui um banco de dados que armazena informações sobre os pacotes instalados no sistema e utiliza essas informações para poder realizar a instalação, atualização e remoção de pacotes de maneira correta.

Configurando o APT

O APT é configurado por meio de diversos arquivos armazenados no diretório */etc/apt/apt.conf.d*. Neste arquivos são definidas opções relativas à configuração de proxy, tempo de timeout para conexões com servidores, entre outras. Outro arquivo de configuração importante é o */etc/apt/sources.list*, onde são definidos os repositórios que serão utilizados para a instalação e atualização de pacotes.

Utilizando o APT

O APT utiliza o comando *apt-get* para realizar diversas tarefas, como instalar, atualizar e remover pacotes. Veremos a seguir as principais funcionalidades deste comando.

Atualizando as lista de pacotes disponíveis

Antes de fazer qualquer instalação, atualização ou remoção de pacotes, é preciso sincronizar o banco de dados do APT com a lista de pacotes disponíveis nos repositórios. Esta ação é executada através do comando abaixo:

```
# apt-get update
```

Instalando pacotes

Para instalar um pacote, basta executar o comando *apt-get* com a opção *install* seguida do nome do pacote. O APT consulta então os repositórios definidos no arquivo *sources.list* em busca da versão mais recente do pacote e em seguida faz o download dele e de suas dependências, caso necessário. Após obter todos os pacotes necessários, a instalação deles é executada. O comando do exemplo seguinte instala o pacote *bind9* e suas dependências:

```
# apt-get install bind9
```

Sempre que utilizamos o *apt-get* para instalar, atualizar ou remover um pacote é apresentado um resumo das ações que serão executadas, solicitando a confirmação do administrador antes de executá-las. O comando do exemplo anterior apresentará o seguinte resumo:

```
Reading package lists... Done
Building dependency tree
```

```
Reading state information... Done
The following extra packages will be installed:
  bind9utils
Suggested packages:
  bind9-doc resolvconf
The following NEW packages will be installed:
  bind9 bind9utils
0 upgraded, 2 newly installed, 0 to remove and 23 not upgraded.
Need to get 466kB of archives.
After this operation, 1,413kB of additional disk space will be used.
Do you want to continue [Y/n]?
```

A opção *install* também pode ser utilizada para atualizar pacotes. A sintaxe do comando para atualizar um pacote é a mesma utilizada para a instalação de novos pacotes. Se o pacote passado como parâmetro não estiver desatualizado, uma mensagem será mostrada informando que a versão instalada já é a mais atual, como mostra o exemplo:

```
# apt-get install wget
Reading package lists... Done
Building dependency tree
Reading state information... Done
wget is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 21 not upgraded.
```

Removendo pacotes

Para remover um pacote e suas dependências, basta utilizar a opção *remove*, como mostra o exemplo:

```
# apt-get remove bind9
```

A opção *remove* só remove os pacotes, mantendo no sistema seus arquivos de configuração. Para removê-los também, é preciso acrescentar a opção *--purge*, como mostra o exemplo:

```
# apt-get remove bind9 --purge
```

Atualizando pacotes

Para atualizar todos os pacotes instalados no sistema, devemos utilizar a opção *upgrade*, como mostra o exemplo:

```
# apt-get upgrade
```

Antes disso, no entanto, é importante atualizar o banco de dados do APT com a opção *update*.

Atualizando a distribuição

Para atualizar toda a distribuição de uma só vez, devemos utilizar a opção *dist-upgrade*, como mostra o exemplo:

```
# apt-get dist-upgrade
```

Limpando o repositório local

Quando instalamos um pacote através do comando *apt-get*, os arquivos necessários são baixados dos repositórios e armazenados em um repositório local, que fica no diretório */var/cache/apt/archives.* Os arquivos utilizados para instalar os pacotes no sistema podem ser removidos com a opção *clean*, como mostra o exemplo abaixo:

```
# apt-get clean
```

Opções mais utilizadas do comando *apt-get*:

- *update* – adquire novas listas de pacotes.
- *install* – instala novos pacotes ou atualiza pacotes desatualizados.
- *remove* – remove um pacote.
- *upgrade* – faz uma atualização.
- *dist-upgrade* – atualiza a distribuição.
- *clean* – apaga arquivos baixados para instalação.

Fazendo buscas e obtendo informações sobre pacotes

É possível fazer buscas por pacotes, consultando a base de dados do APT através do comando *apt-cache* com a opção *search*, como mostra o exemplo a seguir:

```
# apt-cache search dns
```

O comando anterior lista todos os pacotes cujo nome ou descrição casem com o padrão passado como parâmetro na busca. A opção de busca é útil quando não sabemos o nome exato de um pacote. De posse do nome correto do pacote, podemos utilizar a opção *show* do comando *apt-cache* para exibir informações sobre o pacote, como mostra o exemplo abaixo:

```
# apt-cache show dnsutils
```

Dicas sobre gerenciadores de pacotes

- Prefira pacotes feitos para sua distribuição.
- Tome cuidado com arquivos *.rpm* de outras distribuições.
- Arquivos com *headers* estão em pacotes com sufixo "-dev" ou "-devel":
  - *libpng-devel*
- Mantenha o controle dos pacotes que foram instalados a partir do código-fonte.

Prefira sempre instalar aplicativos a partir de pacotes feitos para sua distribuição/versão. Em geral, foram preparados visando a integração com o sistema e já vêm com correções de segurança aplicadas. Ao instalar aplicativos a partir de arquivos-fontes, procure satisfazer os seus pré-requisitos por intermédio de pacotes da própria distribuição. Se for necessário utilize os arquivos de *headers* das bibliotecas, que podem ser encontrados em pacotes especiais da distribuição. Em geral, esses pacotes recebem o nome da biblioteca acrescido dos sufixos "-dev" ou "-devel". Procure manter o código-fonte de aplicações dentro do diretório */usr/src.*

É aconselhável manter um controle dos pacotes que foram instalados a partir do código-fonte ou de arquivos binários, que deverão ser atualizados manualmente sempre que necessário. Em alguns casos, é possível manter todos os arquivos de um aplicativo instalado a partir do código-fonte ou de um arquivo binário sob um único diretório como, por exemplo, */usr/local/<aplicativo>*. Isso pode facilitar a remoção do aplicativo, quando ele não for mais necessário.

Exercício de fixação 2
Gerenciador de pacotes APT

1. Qual a finalidade do comando *apt-get install nome_pacote*?
2. Qual a diferença entre *apt-get upgrade* e *apt-get dist-upgrade*?
3. Qual o gerenciador de pacotes da distribuição Debian?
4. O que faz o comando *apt-cache*?
5. O que faz o comando *apt-cdrom*?
6. Pesquise a diferença entre *aptitude* e *apt-get*.
7. O que é um Mirror?

Roteiro de Atividades 9

Atividade 9.1 – Encontrando bibliotecas utilizadas por um programa

Escolha três arquivos binários de programas dentro do diretório */usr/bin*. Verifique quais bibliotecas esses binários utilizam.

Atividade 9.2 – Instalando uma aplicação a partir do seu código-fonte

Instale a última versão do aplicativo *wget* a partir do seu código-fonte e configure-o para que seja instalado dentro do diretório */usr/local/curso/*. Caso não seja possível instalar, descreva as dificuldades encontradas. Verifique o conteúdo de */usr/local/curso* após a instalação.
O código-fonte do *wget* pode ser obtido na seguinte URL: http://ftp.gnu.org/gnu/wget.

Atividade 9.3 – Instalando uma aplicação a partir de um arquivo binário

Instale o plugin do flash a partir de seu arquivo binário e descreva o que aconteceu na instalação.

Atividade 9.4 – Instalando um pacote *rpm*

A partir do arquivo de instalação, localize o arquivo de pacote do programa tcpdump e o instale ou atualize-o utilizando o comando *rpm*. Descreva o que você fez para instalar o programa e justifique qualquer problema ocorrido.

Se o tcpdump já estiver instalado no sistema, o comando retornará a seguinte mensagem de erro: "o pacote tcpdump-14:4.0.0-3.20090921gitdf3cb4.1.el6.x86_64 já está instalado".

Atividade 9.5 – Descobrindo a qual pacote pertence uma biblioteca

Descubra a que pacote pertencem as bibliotecas utilizadas por um dos binários escolhidos na Atividade 9.1.

Atividade 9.6 – Descobrindo as dependências dos pacotes

Utilizando o comando *rpm*, descubra algumas das dependências do pacote utilizado na atividade anterior. Verifique a utilidade de cada dependência.

Atividade 9.7 – Atualizando o sistema com *yum*

Utilize o *yum* para descobrir se há algum pacote a ser atualizado no seu sistema. Se houver, proceda com a atualização.

Atividade 9.8 – Instalando pacotes com *yum*

Utilize o *yum* para instalar a biblioteca *ncurses* e os seus arquivos de *headers*.

10

Configuração e utilização de dispositivos de hardware

objetivos

Configurar e utilizar os principais tipos de dispositivos, como unidades de CD/DVD, pen drives, placas de rede, controladoras SCSI e placas de vídeo.

conceitos

Arquivos de dispositivos, módulos, gerenciamento e configuração de dispositivos e gerenciamento de energia.

Introdução

Suporte a dispositivos no Linux:

- Como um dispositivo interage com o sistema operacional Linux?

Um driver no Linux pode ser:

- Embutido no kernel.
- Um módulo do kernel.

O Linux, assim como outros sistemas operacionais modernos, é capaz de suportar um grande número de dispositivos de hardware. Particularmente nos últimos anos, grandes mudanças foram feitas no sistema, permitindo um melhor suporte a uma série de dispositivos, tornando-os progressivamente plug-and-play. Embora esteja cada vez mais fácil configurar e utilizar um dispositivo no Linux, seja de forma automática ou com a ajuda de algum aplicativo gráfico, abordaremos alguns conceitos por trás dessas facilidades e, sempre que possível, demonstraremos como configurar e utilizar os principais dispositivos sem fazer uso de aplicativos gráficos. Assim, o aluno estará se preparando para situações em que esses aplicativos não estejam disponíveis (como em um servidor, por exemplo) ou quando a configuração automática eventualmente não funcionar. Veremos como funciona o suporte aos dispositivos no Linux abordando o kernel e seus subsistemas. Em seguida, veremos como configurar e utilizar algumas das classes de dispositivos mais comuns.

Para que um sistema operacional seja capaz de utilizar um dispositiv, é necessário um pequeno programa capaz de se comunicar com o dispositivo em questão, traduzindo requisições do sistema para comandos deste dispositivo. Esse programa é conhecido como driver e, no Linux, está intimamente ligado ao kernel, podendo inclusive estar embutido nele. No entanto, como o número de dispositivos suportados é bastante grande e embutir todos os drivers diretamente no kernel o tornaria grande demais, foi criado um mecanismo que torna

possível separar os drivers em pequenos arquivos chamados de módulos, que podem ser carregados e utilizados pelo kernel sob demanda. Uma vez carregados, esses módulos funcionam como se fossem uma parte do kernel, sendo executados com os mesmos privilégios que ele, o que é conhecido como kernel-space. Usualmente, o kernel é compilado de forma a ter suporte aos principais dispositivos, como teclados, mouses, placas de rede, discos rígido, etc. Os drivers desses dispositivos são embutidos diretamente no kernel e os drivers dos demais dispositivos são disponibilizados como módulos.

Exercício de fixação 1
Verificando o kernel

Qual comando você pode utilizar para verificar a versão do kernel em seu sistema?

Arquivos de dispositivos

Arquivos de dispositivos:

- Permitem operações como: abrir, fechar, ler e escrever;
- Localizados no diretório */dev*.
- Criados através do comando *mknod*.
- Números Major e minor utilizados pelo kernel para identificar o tipo de dispositivo e o dispositivo em si, respectivamente;
- Podem ser orientados poa bloco ou caractere.

Os projetistas do sistema Unix, no qual o Linux se baseia, tinham como princípio de projeto a máxima de que "tudo no Unix é um arquivo". Com isso, pretendiam simplificar o desenvolvimento de aplicações, oferecendo aos programadores uma interface com o sistema onde todos os seus recursos são tratados como arquivos, isto é, suportando as operações de abrir, fechar, ler e escrever. A grande maioria dos dispositivos no Linux está associada a um arquivo especial no diretório */dev* por meio do qual os programas podem se comunicar com estes dispositivos. Por exemplo, se um programador deseja escrever um programa que leia um determinado dado de uma porta serial, basta fazer com que esse programa abra o arquivo */dev/ttyS0* e leia os seus dados. As mesmas permissões aplicadas a arquivos comuns também são aplicadas a arquivos de dispositivos. Sendo assim, é possível controlar qual usuário ou grupo de usuários tem acesso a um dispositivo. Cada arquivo de dispositivo é criado por intermédio do comando *mknod*, e está associado a dois números chamados de *major* e *minor*. Esses números são utilizados pelo kernel para identificar a qual dispositivo estão associados. Além disso, cada dispositivo pode ser do tipo block (quando manipulam dados em blocos de 512 bytes ou múltiplos) ou character (quando manipulam dados byte a byte). O primeiro tipo permite a leitura e a escrita de maneira aleatória (pode-se especificar de onde se quer ler ou escrever). Já no segundo, a leitura e a escrita só podem ser feitas de forma sequencial.

A Tabela 10.1 mostra alguns dispositivos do Linux, com suas descrições e arquivos associados.

Arquivo	Dispositivo	Descrição
hda	Disco ou unidade de CD/DVD IDE.	Disco IDE master conectado à controladora IDE primária.
hda1	Primeira partição primária do disco IDE master conectado à controladora IDE primária.	Os arquivos hda1 até hda4 são as partições primárias de um disco. A partir de hda5 são as partições estendida e lógicas.

Arquivo	Dispositivo	Descrição
hdb	Disco ou unidade de CD/DVD IDE.	Disco IDE slave conectado à controladora IDE primária.
hdc	Disco ou unidade de CD/DVD IDE.	Disco IDE master conectado à controladora IDE secundária.
hdd	Disco ou unidade de CD/DVD IDE.	Disco IDE slave conectado à controladora IDE secundária.
ttySO	Primeira interface serial.	As interfaces seriais são identificadas por ttyS0, ttyS1 etc.
dsp0	Primeira placa de som.	
mixer	Dispositivo de controle da placa de som.	
sda	Disco ou unidade de CD/DVD SCSI ou pen drive.	Assim como os discos IDE, os discos SCSI são identificados por sda, sdb etc. e as partições são identificadas como sda1, sda2 etc.
scd0	Unidade de CD/DVD SCSI.	Primeira unidade de CD/DVD SCSI. Os demais são identificados por scd1, scd2, etc.
input/ mice	Mouse.	Mouse PS2 ou USB.
input/ event0	Teclado.	Teclado PS2.
cdrom	Unidade de CD/DVD IDE.	Link para hda, hdb, hdc ou hdd, pois a unidade de CD/DVD é tratada como um disco IDE.

Tabela 10.1
Principais dispositivos do Linux.

Exercício de fixação 2
Arquivos de dispositivos

Descreva o que seriam os dispositivos abaixo:

- Sda3
- Hda2
- Cdrom
- Scd2
- Dsp2

Módulos

Módulos são pequenos arquivos que contêm trechos de códigos que implementam funcionalidades do kernel. Eles fornecem suporte a dispositivos de hardware ou a funcionalidades do sistema operacional.Os módulos utilizados pelo kernel são específicos para cada versão e se encontram no diretório */lib/modules/<versao_kernel>*. Por meio dos comandos *insmod*, *modprobe*, *rmmod e lsmod* é possível carregar módulos no kernel, remover e listar os módulos em uso. Veremos a seguir como efetuar cada uma dessas operações.

Para listar todos os módulos carregados pelo kernel, basta utilizar o comando *lsmod*, conforme o exemplo abaixo:

```
# lsmod
Module Size Used by
lp 12164 0
nfs 208324 1
lockd 62472 2 nfs
sunrpc 144452 3 nfs,lockd
parport_pc 41540 1
parport 38856 2 lp,parport_pc
```

A saída do comando *lsmod* apresenta o nome dos módulos, o seu tamanho em bytes, o número de instâncias dos módulos que estão carregadas e quais são os outros módulos que os utilizam, já que um módulo pode prover funções necessárias para outro módulo. Observe também que o nome de um módulo é o nome do arquivo que contém esse módulo sem sua extensão.

Para carregar um módulo manualmente, podemos utilizar os comandos *insmod* ou *modprobe*. O comando *insmod* insere apenas o módulo especificado na linha de comando, enquanto o comando *modprobe* é capaz de inserir o módulo especificado e ainda carregar de forma automática os módulos adicionais utilizados pelo módulo especificado. É possível, também por meio desses comandos, passar alguns parâmetros para o módulo que está sendo carregado. A sintaxe desses comandos é apresentada a seguir:

```
# insmod <nome_do_modulo> [parametros]
# modprobe <nome_do_modulo> [parametros]
```

Para obter mais informações sobre um módulo, como por exemplo saber os parâmetros que ele aceita, podemos utilizar o comando *modinfo*. O comando do exemplo abaixo mostra as informações relativas ao módulo *lp*:

```
# modinfo lp
filename: /lib/modules/2.6.13.2/kernel/drivers/char/lp.ko
alias: char-major-6-*
license: GPL
vermagic: 2.6.13.2 SMP preempt PENTIUM4 gcc-4.0
depends: parport
parm: reset:bool
parm: parport:array of charp
```

Para descarregar um módulo da memória, utilizamos o comando *rmmod*. Convém notar que, caso o módulo esteja sendo utilizado por outro módulo, o sistema não irá removê-lo. O comando *modprobe* pode ser utilizado para remover tanto um determinado módulo quanto os módulos que dependem desse módulo utilizando-se a opção *-r*, como no exemplo abaixo:

```
# modprobe -r lp
```

Módulos são específicos para cada versão do kernel.

São encontrados no diretório */lib/modules/<versão do kernel>.*

Comandos relacionados:

- *lsmod* – lista os módulos carregados na memória;
- *insmod* – carrega um modulo;
- *modprobe* – carrega um módulo e seus módulos dependentes;
- *modinfo* – exibe informações sobre um módulo;
- *rmmod* – descarrega um módulo da memória. O comando *modprobe* pode ser utilizado com a opção *-r* para descarregar um módulo e seus módulos dependentes.

Exercício de fixação 3
Módulos

- Liste alguns módulos carregados na memória.
- Carregue algum módulo.
- Exibir informações sobre algum módulo.
- Descarregue o módulo anteriormente carregado.

Initrd

Imagine que o sistema de arquivos em que se encontram os módulos do kernel esteja armazenado em um disco conectado a uma controladora de discos que necessita de um módulo do kernel para poder ser utilizada pelo sistema operacional. Temos, então, um impasse. Para resolver este problema, os programadores do kernel criaram o initrd (initial RAM disk).

O initrd é um sistema de arquivos temporário, que é carregado em memória RAM durante o boot do sistema operacional, no qual estão presentes todos os módulos que o kernel precisa no momento do boot. O initrd é carregado juntamente com o kernel pelo carregador de boot de segundo estágio.

Como ter acesso a um disco SCSI durante o boot se o driver para esse dispositivo está em um módulo do kernel?

O kernel precisa do módulo para acessar o disco SCSI, mas esse módulo está armazenado no disco SCSI e o kernel ainda não carregou o módulo da controladora de discos SCSI.

Solução: sistema de arquivos simples, que o carregador de boot entende, contendo os módulos necessários para o sistema inicializar.

Gerenciando dispositivos

O gerenciamento de dispositivos no Linux é feito de forma automática por programas que serão descritos neste tópico.

Hotplug

Desde a introdução da série 2.6 do kernel do Linux, este passou a contar com um subsistema conhecido como hotplug, cuja função é detectar e gerar eventos sempre que um novo dispositivo é conectado a um barramento USB, PCI, SCSI, PCMCIA ou firewire. O hotplug

também é capaz de identificar os dispositivos presentes durante o processo de boot do sistema. Isto é feito através de mensagens enviadas aos barramentos citados. Quando algum dispositivo é conectado, ele responde à mensagem fornecendo ao kernel um identificador (ID). Com esse ID, o kernel pode consultar uma tabela para identificar o dispositivo e executar em espaço de usuário um programa também chamado de hotplug, que é capaz de carregar os módulos de controle de um dispositivo e configurá-lo automaticamente.

Além disso, o hotplug também pode executar scripts localizados nos diretórios */etc/hotplug* e */etc/hotplug.d*. Um script pode, por exemplo, criar um ícone no desktop para acesso a um pen drive, sempre que este for conectado ao computador. Neste módulo, não veremos a criação de scripts para atuar em conjunto com o hotplug.

Udev

Tradicionalmente, o conteúdo do diretório */dev* era estático, ou seja, durante a instalação, um script criava todos os arquivos de dispositivos, mesmo que seus dispositivos associados não estivessem instalados no sistema. Além disso, se fosse instalado um novo tipo de dispositivo para o qual não existisse um arquivo de dispositivo correspondente, era necessário criar esse arquivo manualmente. Para evitar essa situação, foi criado um sistema de arquivos conhecido como devfs, capaz de criar esses dispositivos automaticamente. Posteriormente, esse sistema foi substituído por outro, conhecido como udev, que inicialmente foi concebido para trabalhar em conjunto com o hotplug. Assim, quando um dispositivo era detectado, o kernel disparava um processo também chamado de hotplug, que, por sua vez, acionava o udev para que ele criasse o arquivo de dispositivo dinamicamente. Em algumas distribuições, o udev trabalha em conjunto com o hotplug, enquanto em outras substitui completamente o hotplug, executando suas funções.

Uma característica do udev é que podemos definir arquivos de dispositivos fixos para cada dispositivo de hardware utilizado. Isto facilita a criação de scripts utilizados pelo udev, pois caso tenhamos dois dispositivos do mesmo tipo, como por exemplo duas impressoras, pode acontecer de, a cada boot do sistema, elas serem associadas a arquivos de dispositivos diferentes, o que dificultaria a automatização de tarefas associadas a cada uma das impressoras nesses scripts. Por exemplo, ao invés de utilizarmos os arquivos de dispositivo padrão para impressoras, */dev/lp0 e /dev/lp1*, podemos definir através do udev os arquivos */dev/hp* e */dev/epson*. Assim, nos scripts disparados pelo udev, as ações executadas pela impressora HP estariam sempre associadas ao arquivo de dispositivo */dev/hp*.

Outra característica importante do udev é o uso de regras que permitem a execução de ações quando um dispositivo é conectado ao sistema. As regras do udev ficam localizadas em arquivos dentro do diretório */etc/udev/rules.d* e possuem uma sintaxe específica. Um exemplo de regra, pode ser abrir o gerenciador de arquivos konqueror, sempre que um pen drive for conectado. A regra do exemplo abaixo executa o script *connect_pendrive.sh* sempre que um pen drive for conectado ao computador.

```
BUS="usb", ACTION=="add", KERNEL=="sd??", NAME="%k", RUN+="/usr/
local/bin/connect_pendrive.sh"
```

A sintaxe das regras do udev é bem intuitiva, como pode ser visto no exemplo acima, mas não será apresentada neste módulo.

- Detecção de dispositivos no Linux.
- Plug and Play.
- O suporte é dado pelo kernel por intermédio do hotplug:
  - ID do dispositivo;
  - Verifica o ID em uma tabela.
- Módulo pode ser carregado automaticamente com o auxílio do programa hotplug.

O diretório */dev* passou a ser criado em memória, contendo apenas os arquivos de dispositivos que realmente estavam conectados ao sistema. Nas versões do kernel posteriores à versão 2.6.15, o programa hotplug passou a ser substituído por uma nova versão do udev, capaz de fazer todas as funções anteriormente desempenhadas pelos dois programas, isto é, carregar módulos, configurar o dispositivo e criar os arquivos de dispositivo.

O conteúdo do diretório */dev* é estático.

Desvantagens:

- Centenas de arquivos de dispositivos eram criados, mas muitos jamais seriam utilizados.
- Um novo tipo de dispositivo poderia necessitar que um arquivo de dispositivo fosse criado manualmente.

Soluções:

- devfs
- udev (evolução do devfs, com suporte às facilidades do hotplug).

Exercício de fixação 4
Gerenciando dispositivos

Descreva a diferença entre hotPlug e udev.

Identificando e configurando dispositivos

Conforme foi visto, cada vez mais as novas versões de kernel e programas associados se encarregam de fazer todo o trabalho de identificação e configuração de dispositivos.
No entanto, em alguns poucos casos, ainda é necessária a intervenção do administrador para configurar determinados dispositivos. Na maioria dos casos, isto se deve ao fato de que os módulos necessários para suportar estses dispositivos não estão disponíveis no sistema. Algumas ferramentas podem ser bastante úteis nessa tarefa, como veremos a seguir.

Para identificar os dispositivos PCI conectados, podemos utilizar o comando *lspci*, que lê informações de arquivos localizados no diretório */proc* e as apresenta em um formato mais apropriado. O diretório */proc* é um sistema de arquivos virtual criado pelo kernel durante o boot, que armazena seus dados em memória RAM.

O comando *lspci* na sua forma mais básica traz a posição do dispositivo no barramento, seguida de sua descrição, como mostra o exemplo abaixo:

```
# lspci
0000:00:00.0 Host bridge: Silicon Integrated Systems [SiS]
740 Host (rev 01)
0000:00:01.0 PCI bridge: Silicon Integrated Systems [SiS]
```

```
Virtual PCI-to-PCI bridge (AGP)
0000:00:02.5 IDE interface: Silicon Integrated Systems
[SiS] 5513 [IDE]
0000:00:02.7 Multimedia audio controller: Silicon
Integrated Systems [SiS] Sound Controller (rev a0)
0000:00:03.0 USB Controller: Silicon Integrated Systems
[SiS] USB 1.0 Controller (rev 0f)
0000:00:03.3 USB Controller: Silicon Integrated Systems
[SiS] USB 2.0 Controller
0000:00:04.0 Ethernet controller: Silicon Integrated
Systems [SiS] SiS900 PCI Fast Ethernet (rev 90)
0000:01:00.0 VGA compatible controller: Silicon Integrated
Systems [SiS] 65x/M650/740 PCI/AGP VGA Display Adapter
```

No exemplo acima, podemos ver que o comando *lspci* lista também um dispositivo AGP (a placa de vídeo), os controladores USB e alguns subsistemas da placa-mãe, como o controlador IDE e o Host Bridge.

Apesar das facilidades do hotplug, algumas vezes é necessário que o administrador do sistema identifique algum dispositivo manualmente.

Para listar os dispositivos conectados ao barramento PCI:

- O comando *lspci* lê dados do diretório */proc/bus/pci* e apresenta os dispositivos instalados.

Outros comandos úteis para verificação de informações sobre os dispositivos instalados são listados a seguir:

- **lscpu** – exibe diversos parâmetros da CPU, que são obtidos através do arquivo */proc/cpuinfo*.
- **lshw** – exibe informações detalhadas a respeito do hardware instalado no computador. Entre as informações exibidas, podemos destacar: dados sobre a placa-mãe, CPU, memória, barramentos, interfaces, discos, entre outros.
- **lsusb** – exibe informações sobre os barramentos USB disponíveis no sistema e sobre os dispositivos a eles conectados.

O diretório */proc* é também uma fonte de informações sobre o hardware instalado no computador. Nele temos diversos arquivos, entre os quais podemos destacar:

- **/proc/cpuinfo** – arquivo que exibe informações sobre a CPU.
- **/proc/meminfo** – arquivo que exibe informações sobre a memória.
- **/proc/devices** – arquivo que exibe informações sobre dispositivos ativos no sistema.
- **/proc/filesystems** – arquivo que exibe a lista de sistemas de arquivos suportados pelo kernel.
- **/proc/interrupts** – arquivo que exibe a lista de interrupções e seus respectivos dispositivos.
- **/proc/acpi** – diretório que contém informações sobre o sistema de gerenciamento de energia Advanced Configuration and Power Interface (ACPI).

- **/proc/bus** – diretório que contém informações sobre dispositivos PCI, USB, entre outros.
- **/proc/net** – diretório que contém informações sobre os protocolos de rede disponíveis no sistema.
- **/proc/sys** – diretório que, além de fornecer informações sobre o sistema, permite que parâmetros de configuração do kernel sejam alterados.

Além de fornecer diversas informações sobre o hardware do sistema, ainda é possível configurarmos diversos parâmetros do kernel através do diretório */proc*. No entanto, neste curso não veremos a parte de configuração do kernel.

O diretório */proc* contém diversas informações sobre o sistema, com destaque para os arquivos:

- /proc/cpuinfo
- /proc/meminfo
- /proc/devices
- /proc/filesystems
- /proc/interrupts
- /proc/acpi
- /proc/bus
- /proc/net
- /proc/sys

Os kernels a partir da versão 2.6 apresentam também o diretório */sys*, que contém informações sobre dispositivos de maneira análoga ao diretório */proc*.

Unidades de CD/DVD

Unidades de CD/DVD IDE:

- Barramento primário:
  - Master *hda*
  - Slave *hdb*
- Barramento secundário:
  - Master *hdc*
  - Slave *hdd*

Unidades de CD/DVD SCSI:

- Primeiro dispositivo detectado: scd0
- Segundo dispositivo detectado: scd1

O dispositivo é configurado durante a instalação:

- Arquivo */etc/fstab*

Unidades de CD/DVD do tipo IDE são suportadas de maneira praticamente transparente pelo kernel. Quando conectamos uma unidade de CD/DVD ao computador, esta é associada a um arquivo de dispositivo de acordo com a sua posição no barramento IDE. Se a unidade estiver conectada ao barramento primário, será associada aos arquivos *hda* ou *hdb*, caso esteja configurada como master ou slave, respectivamente. Já se estiver conectada ao barramento secundário, será associada aos arquivos *hdc* ou *hdd*, caso esteja configurada como master ou slave, respectivamente.

Unidades de CD/DVD do tipo SCSI necessitam que a interface SCSI, onde estão conectados, tenha sido corretamente reconhecida, e que o módulo que suporta unidades de CD/DVD SCSI tenha sido carregado, o que, normalmente, acontece de forma automática. Uma vez reconhecidas, as unidades de CD/DVD SCSI são associadas aos seus arquivos de dispositivos: *scd0* para a primeira unidade, *scd1* para a segunda e assim por diante.
Durante a instalação do Linux, o programa instalador identifica em que tipo de barramento e em que posição a unidade de CD/DVD se encontra e configura o arquivo */etc/fstab* de acordo com essa localização.

O programa instalador pode também criar o link */dev/cdrom*, apontando para o arquivo real correspondente à unidade de CD/DVD. O arquivo */etc/fstab* é lido por alguns scripts de inicialização durante o boot, e pelos comandos *mount* e *umount*. Neste arquivo são relacionados os arquivos de dispositivos, o seu ponto de montagem, o tipo de sistema de arquivos, as opções de montagem, a opção de dump e a ordem de verificação do sistema de arquivos. Para obter mais detalhes sobre o arquivo */etc/fstab*, sua página de manual pode ser consultada.

Durante o boot, o kernel identifica os dispositivos conectados, listando-os um a um. Estas informações podem ser recuperadas com o comando *dmesg*. O exemplo abaixo mostra o trecho que lista a unidade de CD/DVD IDE.

```
...
Probing IDE interface IDE0...
hdb: HL-DT-ST DVDRAM GSA-4163B, ATAPI CD/DVD-ROM drive
hdb: ATAPI 40X DVD-ROM DVD-R-RAM CD-R/RW drive, 2048kB Cache,
UDMA(33)
```

O arquivo de dispositivo pode ser um link:

- */dev/cdrom*

O comando *dmesg* pode ajudar a identificar qual o dispositivo correspondente à unidade de CD/DVD.

```
...
Probing IDE interface IDE0...
hdb: HL-DT-ST DVDRAM GSA-4163B, ATAPI CD/DVD-ROM drive
hdb: ATAPI 40X DVD-ROM DVD-R-RAM CD-R/RW drive, 2048kB Cache,
UDMA(33)
```

Observe que, no trecho acima, estão listados um disco IDE (hda) e suas partições, e o drive de CD/DVD (hdb). Ao contrário de discos IDE, CDs não apresentam partições, portanto, são acessados por intermédio de seu arquivo raiz, que no exemplo acima seria */dev/hdb*. Assim como ocorre com discos IDE, para podermos ler o conteúdo de um CD ou DVD, temos que montá-lo antes, utilizando o comando mount. Supondo que a unidade de CD/DVD esteja associada ao arquivo */dev/hdb*, poderíamos montá-lo no diretório */mnt/cdrom* com o seguinte comando:

```
# mount /dev/hdb /mnt/cdrom
```

Poderíamos alterar o arquivo */etc/fstab*, introduzindo a linha abaixo, para que a unidade de CD/DVD seja montada automaticamente no boot.

```
/dev/hdb /mnt/cdrom iso9660 ro,user,noauto 0 0
```

Nesta linha, indicamos o dispositivo, o seu ponto de montagem, o tipo de sistema de arquivos (CDs e DVDs utilizam sempre o tipo iso9660), seguido das opções de montagem. No caso de CDs e DVDs, as opções mais utilizadas são: ro (read only), noauto (o dispositivo não é montado automaticamente durante o boot) e user (permite que qualquer usuário monte o dispositivo e não apenas o usuário root). Com o arquivo */etc/fstab* configurado adequadamente, um usuário do sistema poderia montar a unidade de CD/DVD indicando apenas o ponto de montagem, como mostra o exemplo abaixo:

```
$ mount /mnt/cdrom
```

Para ejetarmos um CD ou DVD, devemos desmontar a unidade antes, utilizando o comando *umount*, como mostra o exemplo abaixo:

```
$ umount /mnt/cdrom
```

É importante ressaltar que o Linux não permite que um dispositivo seja desmontado se algum arquivo contido nele estiver sendo utilizado por algum recurso do sistema. Assim, o simples fato de existir um shell aberto com o diretório corrente pertencente ao dispositivo impede que o mesmo seja desmontado. Além disso, o botão eject da unidade de CD/DVD não funciona enquanto o dispositivo estiver montado.

Montando uma unidade de CD/DVD:

```
mount /dev/hdb /mnt/cdrom
```

Configurar o *fstab* para facilitar o uso:

```
/dev/hdb /mnt/cdrom iso9660 ro,user,noauto  0  0
```

Com isso, temos o uso simplificado por parte de qualquer usuário:

```
mount /mnt/cdrom
```

Após o uso do CD ou DVD, a unidade deve ser desmontada com o comando abaixo:

```
umount /mnt/cdrom
```

Para desmontar a unidade de CD/DVD, todos os arquivos do dispositivo devem estar fechados. Além disso, não é possível utilizar o botão eject se o dispositivo estiver montado.

Dispositivos de armazenamento USB

O dispositivo de armazenamento USB é suportado pelo kernel por meio do módulo usb-storage, que, por sua vez, utiliza subsistema SCSI.

Pen drives são tratados como discos SCSI:

- /dev/sda1
- /dev/sda2

Os dispositivos de armazenamento USB são suportados por meio do módulo usb-storage que, por sua vez, utiliza o subsistema SCSI. Assim, os dispositivos de armazenamento USB como pen drives são tratados como se fossem discos SCSI, utilizando como arquivos de dispositivos os tradicionais */dev/sdXY* (onde X é uma letra e Y um número). Ao conectarmos um pen drive, o subsistema hotplug detecta este evento, identifica o dispositivo, carrega os módulos necessários e monta o dispositivo. O processo de montagem é semelhante ao de unidades CD/DVD, mudando apenas o arquivo de dispositivo associado. Podemos verificar se o sistema reconheceu o pen drive executando o comando *dmesg*. Em seguida, podemos

verificar qual arquivo de dispositivo foi designado para o pen drive, mediante as informações obtidas no arquivo */proc/partitions*. Este arquivo lista os discos e partições encontrados no sistema, como mostra o exemplo abaixo:

```
# cat /proc/partitions

Major Minor Blocks Name

3 0 58605120 hda

3 1 25438896 hda1

3 2 594405 hda2

8 0 1993728 sda

8 1 1993724 sda1
```

No exemplo acima, podemos observar que a máquina possui um disco IDE com duas partições e um pen drive de 2 GB com uma partição.

Os pen drives geralmente utilizam o sistema de arquivos VFAT para manter a compatibilidade com sistemas Windows, mas nada nos impede de criar nelas um sistema de arquivos do tipo ext3 ou ext4.

- O hotplug pode carregar automaticamente os módulos necessários para o uso de pen drives.
- O comando *dmesg* pode informar se o sistema reconheceu o dispositivo.
- O arquivo */proc/partitions* mostra qual o arquivo de dispositivo está associado ao pen drive.

```
# cat /proc/partitions
Major            Minor         Blocks Name
3  0             58605120      hda
3  1             25438896      hda1
3  2             594405 hda2
8  0             1993728sda
8  1             1993724sda1
```

Interfaces de rede

- Interfaces de rede possuem um bom suporte no Linux.
- Não possuem um arquivo de dispositivo associado no diretório */dev*.
- O acesso a esses dispositivos se faz por meio de sockets.
- Interfaces ethernet são usualmente identificadas pelo sistema como ethX.
- Interfaces wireless são usualmente identificadas pelo sistema como wlanX.

Interfaces de rede são, provavelmente, um dos dispositivos mais bem suportados pelo Linux. Desde o seu surgimento até hoje, a Internet é o principal meio de comunicação entre os desenvolvedores do sistema, logo, é natural que seja dada uma atenção especial para os dispositivos de rede. Usualmente, as interfaces de rede são detectadas pelo subsistema *hotplug/udev* durante o boot, e seus módulos são automaticamente carregados. Ao contrário de outros dispo-

sitivos, as interfaces de rede não possuem um arquivo de dispositivo a elas associado. O acesso a este tipo de dispositivo acontece por outro método, conhecido como sockets.

Usualmente, as interfaces de rede que utilizam o padrão ethernet são nomeadas como ethX, onde X é um número inteiro. Já as interfaces de rede sem fio são usualmente nomeadas como wlanX, embora também possam ser nomeadas utilizando o mesmo padrão das interfaces ethernet.

Exibindo as interfaces de rede do sistema

Para verificarmos as interfaces de rede disponíveis no sistema, podemos listar conteúdo do arquivo */proc/net/dev* ou utilizar o comando *ip*, como mostra o exemplo abaixo:

```
# ip link list

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast
state UP qlen 1000

    link/ether 08:00:27:de:d1:99 brd ff:ff:ff:ff:ff:ff
```

Observe que são listados dois dispositivos de redes (*lo* e *eth0*). O dispositivo *lo* refere-se à interface de loopback, que é um dispositivo virtual utilizado para comunicação local entre aplicações. Logo, o sistema acima possui apenas uma placa de rede, identificada pelo nome eth0.

O comando acima mostra ainda o endereço físico (MAC Address) das interfaces listadas. Este e outros comandos de configuração de interfaces de rede serão vistos em detalhes em outro módulo. Veremos aqui apenas como fazer configurações simples de rede.

- O arquivo */proc/net/dev* contém os dispositivos de redes disponíveis no sistema.
- O comando *ip link list* também retorna dados dos dispositivos de redes do sistema.

```
# ip link list

1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue

    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000

    link/ether 00:02:55:5d:07:c6 brd ff:ff:ff:ff:ff:ff
```

Gerenciando as interfaces de rede do sistema

O comando do exemplo abaixo ativa a interface de rede eth0, utilizando o endereço IP 192.168.0.1 e a máscara de rede 255.255.255.0:

```
# ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
```

Para desativarmos a interface eth0, podemos utilizar também o comando *ifconfig* como mostra o exemplo abaixo:

```
# ifconfig eth0 down
```

Para tornar estas modificações permanentes, devemos editar os arquivos de configuração *ifcfg-ethX* dentro do diretório */etc/sysconfig/network-scripts*, o que pode ser feito manualmente ou com o auxílio de um programa de configuração gráfico, tópico que também não será abordado neste módulo.

O principal comando para ativar e modificar endereços de redes é o *ifconfig*:

```
# ifconfig eth0 192.168.0.1 netmask 255.255.255.0 up
# ifconfig eth0 down
```

Configurações permanentes:

```
/etc/sysconfig/network-scripts/ifcfg-ethX
```

Placas SCSI

- O padrão Small Computer System Interface (SCSI) estabelece um conjunto de interfaces e comandos para transferência de dados entre dispositivos.
- O Linux oferece suporte a vários tipos de controladoras SCSI.
- A ordem de detecção é dada pelo ID do disco no barramento.
- Discos SCSI também podem ser montados e utilizados da mesma maneira que discos IDE.
- Eventualmente, torna-se necessário o uso de initrd para carregar os módulos SCSI da controladora durante o processo de boot do sistema.

O padrão SCSI estabelece um conjunto de interfaces e comandos para transferência de dados entre dispositivos. Este padrão geralmente é utilizado em servidores, que necessitam de discos de alto desempenho. Além disso, o preço de dispositivos SCSI não os torna atraentes para uso em desktops.

Para que um disco SCSI funcione, ele deve ser conectado a uma controladora de discos compatível com o padrão SCSI. O Linux suporta vários tipos de placas controladoras SCSI e o subsistema hotplug é capaz de detectar e carregar o módulo apropriado para cada controladora.

Ao contrário de discos IDE, em sistemas SCSI não existe o conceito de controladoras primária e secundária e discos master e slave. Um disco em um barramento SCSI é nomeado de acordo com a ordem na qual é detectado. Assim, o primeiro disco SCSI é associado ao dispositivo */dev/sda*; o segundo ao */dev/sdb*, e assim por diante. A ordem de detecção é dada pelo ID do disco no barramento (uma configuração feita por meio de jumpers no disco).

Discos SCSI também podem ser montados e utilizados da mesma maneira que discos IDE. A única complicação que aparece fica por conta da situação em que temos o nosso sistema de arquivos montado em um disco SCSI. Neste caso, pode ser necessário acrescentar numa imagem initrd os módulos de suporte à controladora SCSI. Este procedimento não será descrito aqui, mas o aluno que desejar mais informações pode começar consultando o manual de sistema do comando *mkinitrd*.

Placas de vídeo

O Linux possui suporte a placas de vídeo em três modos:

- Modo texto (suporte nativo).
- Modo framebuffer (o suporte deve ser habilitado no kernel).
- Modo gráfico (para suportar a interface gráfica é necessário habilitar o servidor Xorg).

O Linux pode suportar placas de vídeo em três modos distintos: texto, framebuffer e gráfico. O modo texto suporta apenas texto, podendo utilizar também algumas cores. É neste modo que utilizamos a interface de linha de comando. O suporte para o modo texto é nativo no kernel. O modo framebuffer é uma espécie de melhoria do modo texto, onde os programas

acessam diretamente a memória de vídeo da placa, o que possibilita que apresentem imagens e gráficos no que seria apenas um terminal com suporte a texto. Usualmente, esse modo é utilizado pelos kernels das distribuições Linux para a criação de processos de boots gráficos. Esse modo exige o suporte no kernel para uma placa específica ou um padrão de vídeo conhecido como VESA. O modo gráfico, ao qual daremos mais destaque, necessita que a placa de vídeo seja configurada de modo adequado, para que seja suportada pelo servidor Xorg.

A princípio, o Xorg não necessita de nenhum módulo do kernel, já que possui seus próprios módulos. Os módulos do Xorg se encontram no diretório */usr/X11R6/lib/modules/drivers* ou no diretório */usr/lib/xorg/modules/drivers*, e são nomeados utilizando o padrão <nome_do_módulo>_drv.so. Os módulos do Xorg são bastante genéricos. O módulo *trident*, por exemplo, suporta várias placas com chipsets da Trident. Para obter a lista de todas as placas suportadas por um módulo, basta consultar sua página de manual no sistema.

É possível utilizar o comando *Xorg* para gerar automaticamente um arquivo de configuração, como mostra o exemplo abaixo:

```
# Xorg -configure
```

Este comando irá gerar um arquivo de configuração de exemplo, chamado *xorg.conf.new* no diretório */root*, que depois deverá ser copiado para o diretório */etc/X11,* com o nome de *xorg.conf*. O módulo configurado para a placa de vídeo detectada pode ser encontrado na seção device do arquivo *xorg.conf*. Neste módulo não veremos a parte de configuração do Xorg.

O Xorg possui os seus próprios módulos:

- /usr/X11R6/lib/modules/drivers
- /usr/lib/xorg/modules/drivers

Nome dos arquivos de drivers:

- <nome_do_módulo>_drv.so

Um módulo pode suportar diversos modelos de placas do mesmo fabricante.
Configuração automática do Xorg:

- Xorg -configure

A grande maioria das placas também funciona com o módulo VESA, que possui suporte a um conjunto mínimo de funcionalidades definidas pelo padrão VESA. Algumas placas, no entanto, necessitam de um suporte extra do kernel para que o sistema seja capaz de utilizar algumas funcionalidades avançadas disponibilizadas pelo hardware, como aceleração de vídeo e funções 3D. Algumas placas com chipsets da ATI e da NVIDIA necessitam ainda de um módulo especial para o Xorg e outro para o kernel. Estes módulos especiais são disponibilizados pelo fabricante na forma de arquivos de instalação binários.

Interface gráfica Xorg tem funcionalidades avançadas de algumas placas dependem do kernel:

- Aceleração de vídeo.
- Funções 3D.

Placas NVIDIA e ATI possuem um módulo para o Xorg e outro para o kernel (este último geralmente disponibilizado pelo fabricante em modo binário).

Gerenciamento de energia

Praticamente todos os componentes de hardware possuem funcionalidades que visam diminuir o consumo de energia em situações onde estão ociosos. Para que estas funcionalidades possam ser utilizadas é preciso que algum sistema de gerenciamento de energia seja utilizado. O Linux utiliza basicamente dois padrões de gerenciamento de energia, que serão descritos a seguir.

Advanced Power Management (APM)

- API desenvolvida pela Intel e pela Microsoft, que possibilita ao sistema operacional gerenciar o consumo de energia através de funcionalidades do BIOS.
- Com o surgimento do padrão ACPI, o APM foi descontinuado e o suporte no Linux só foi mantido até a versão 2.6.39 do kernel.

O APM é uma API (Application Program Interface) desenvolvida pela Intel e pela Microsoft, que possibilita ao sistema operacional gerenciar o consumo de energia através de funcionalidades do BIOS. O principal objetivo do APM é diminuir o consumo de energia do computador, executando ações como redução do clock da CPU, desativação de discos e do monitor de vídeo, entre outras. O Linux implementa o APM através do daemon apmd, mas é necessário também que o suporte ao APM seja habilitado no kernel. Com o surgimento do padrão ACPI, o APM foi descontinuado e o suporte no Linux só foi mantido até a versão 2.6.39 do kernel.

Advanced Configuration and Power Interface (ACPI)

- Padrão criado pela Intel, Microsoft e Toshiba, que permite gerenciar o consumo de energia dos dispositivos de hardware de um computador.
- Camada do sistema operacional que permite o controle de funcionalidades implementadas. nos dispositivos de hardware, visando controlar o consumo de energia de forma racional.
- É configurado para executar ações quando ocorrem determinados eventos.

O ACPI é um padrão criado pela Intel, Microsoft e Toshiba, que permite gerenciar o consumo de energia dos dispositivos de hardware de um computador. O ACPI é implementado como uma camada do sistema operacional, que permite o controle de funcionalidades implementadas nos dispositivos de hardware, visando controlar o consumo de energia de forma racional. O ACPI, por possuir mais recursos e ser um padrão mais novo que APM, deve ser preferencialmente ativado no sistema, apesar de os dois poderem ser utilizados em conjunto.

O ACPI é implementado no Linux pelo daemon acpid, que é configurado para executar ações quando ocorrem determinados eventos. As ações que são executadas pelo acpid são definidas em arquivos dentro do diretório */etc/acpi/actions* e os eventos são definidos em arquivos dentro do diretório */etc/acpi/events*. O exemplo abaixo mostra o conteúdo do arquivo */etc/acpi/events/powerbtn*, evento que é disparado quando o botão de power é pressionado.

```
event=button[ /]power

action=/etc/acpi/actions/powerbtn.sh
```

Como pode ser visto, o script */etc/acpi/actions/powerbtn.sh* é executado sempre que o evento associado ao botão de power for detectado. O conteúdo deste script não será exibido aqui devido ao seu tamanho, mas sua função é desligar o sistema operacional de forma correta para que dados em memória não sejam perdidos ou corrompidos.

Roteiro de Atividades 10

Atividade 10.1 – Descobrindo os dispositivos detectados pelo kernel

Durante o boot de um sistema Linux, o kernel automaticamente detecta uma série de dispositivos. Você seria capaz, utilizando o comando *dmesg*, de identificar os discos IDE presentes no seu sistema (HDs e unidades de CD/DVD)?

Atividade 10.2 – Verificando os módulos carregados

Verifique os módulos carregados no sistema, selecione três módulos e obtenha mais informações sobre eles.

Atividade 10.3 – Identificando os dispositivos PCI

Identifique os dispositivos PCI conectados ao seu computador. Qual é a placa de vídeo? E a placa de rede?

Atividade 10.4 – Utilizando um pen drive

Nesta atividade o aluno deve solicitar ao instrutor um pen drive.

1. Antes de inserir o pen drive, verifique o conteúdo do arquivo */proc/partitions*.
2. Insira o pen drive e, utilizando o comando *dmesg*, verifique se o kernel o detectou oedispositivo.
3. Verifique novamente o conteúdo do arquivo */proc/partitions*.
4. Monte o pen drive e acesse o seu conteúdo.
5. Desmonte o pen drive.

Atividade 10.5 – Verificando e identificando as placas de rede

Verifique os dispositivos de rede instalados no seu sistema. Quantas placas Ethernet o sistema possui?

Atividade 10.6 – Identificando a placa gráfica e seu driver

Esta atividade permite a detecção e a identificação da placa de vídeo e a verificação do driver usado pelo ambiente gráfico. Ela pode não funcionar corretamentemnum ambiente virtual.

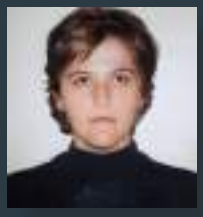
Certifique-se de que o ambiente gráfico não está sendo executado. Talvez seja necessário reiniciar o computador utilizando o comando *telinit 3*. Como usuário root, execute o comando *X –configure*, que gerará um arquivo de configuração no diretório */root*. Identifique na sessão *Device* o driver de vídeo que foi configurado.

Bibliografia

- MEADORS, Todd. *Guide to Linux Shell Script Programing*. Thompson Course Technology. 2003.
- TANEMBAUM, Andrew S. *Sistemas Operacionais Modernos*. 3ª Edição. Pearson. 2010.
- JARGAS, Aurélio Marinho. *Shell Script Profissional*. Novatec. 2008.
- GITE, Vivek G. *Linux Shell Scripting Tutorial*. *A beginner's handbook*. Disponível em: <http://www.dis.uniroma1.it/~bordino/shell-tutorial.pdf>. Acessado em abril 2012.
- KOCHAN, Stephen G.; WOOD, Patrick H. *Exploring the Unix System*, SAMS. Prentice Hall, 1996.
- JANG, Michael. *Dominando Red Hat Linux 9*. Rio de Janeiro: Editora Ciência Moderna, 2003.
- PEEK, Jerry, *et alli*. *Learning Unix Operating System*. O'Reilly & Associates, 2002.
- KDE Project. http://www.kde.org/
- Red Hat, Inc., Fedora Project. http://fedora.redhat.com/
- Red Hat, Inc., Red Hat Linux 9: Red Hat Linux Getting Started Guide. http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/getting-started-guide/
- Red Hat, Inc., Red Hat Linux 9: Red Hat Linux x86 Installation Guide. http://www.redhat.com/docs/manuals/linux/RHL-9-Manual/install-guide/
- The XFree86 Project, Inc., XFree86(TM). http://www.xfree86.org/
- TLDP, The Linux Documentation Project. http://www.tldp.org/
- Informações em português sobre Linux. http://br-linux.org/

- Índice das mais diversas distribuições Linux.
  http://www.linux.org/
- Informações em inglês sobre Linux.
  http://www.linuxjournal.com/
  http://www.linuxcentral.com/_v3/
- Índice de listas de hardware compatível.
  http://www.linux-drivers.org/
- Free Software Foundation.
  http://www.fsf.org/
- Desenvolvimento de software livre.
  http://sourceforge.net/
- Índice de atualizações de software Unix/Linux.
  http://freshmeat.net/
- Comunidade/Fórum brasileiro de Linux.
  http://www.vivaolinux.com.br/
- Sistema Operacional Livre GNU Unix Compatível.
  http://www.gnu.org/
- Licença GPL Versão 3.
  http://gplv3.fsf.org/
- Fabricante do VMware.
  http://www.vmware.com/
- Home page do desktop KDE.
  http://www.kde.org
- Home page do desktop Gnome.
  http://www.gnome.org
- Documentação Linux em geral.
  http://tldp.org/
- The Gnu Operating System – Repositório de software livre.
  http://www.gnu.org
- Fabricante do flash player para o Mozilla.
  http://www.adobe.com
- Fabricante do navegador Mozilla.
  http://www.mozilla.com
- The Linux Documentation Projetc.
  http://www.tldp.org/
- X.Org Foundation.
  http://www.x.org

Arthur Mendes Peixoto possui mais de 26 anos de experiência na área de Redes de Comunicação de Dados e Engenharia de Sistemas, com Dissertação de Mestrado em "Análise de Performance de Sistemas Distribuídos", no Instituto Militar de Engenharia - IME. Participou do desenvolvimento e implantação das primeiras redes com tecnologias ATM, Frame Relay, IP/ MPLS. Foi consultor de grandes projetos como o Backbone IP do Plano Nacional de Banda Larga (PNBL) da Telebrás. Trabalhou por 22 anos no setor de Telecomunicações da Embratel, atuando na prospecção de novas tecnologias para as Redes de Nova Geração – NGN. Participou de testes e verificações de requisitos de RFPs, nos países: EUA, Canadá, Japão, França, Espanha e México. Atuou no desenvolvimento de sistemas no CPqD - Campinas (SP).



Andreia Gentil Bonfante possui graduação em Bacharelado em Ciências de Computação pela Universidade Estadual de Londrina, mestrado e doutorado em Ciências da Computação e Matemática Computacional pela Universidade de São Paulo. Atualmente é professora/pesquisadora da Universidade Federal de Mato Grosso. Tem experiência na área de Ciência da Computação, com ênfase em Inteligência Artificial, atuando principalmente nos seguintes temas: Processamento de Língua Natural, Mineração de Textos e Aprendizado de Máquina. Atua na Educação a Distância como Coordenadora da Especialização em Informática na Educação. Atuou também como instrutora dos cursos de Introdução ao Linux da Escola Superior de Redes na Unidade de Cuiabá.

esr.rnp.br

LIVRO DE APOIO AO CURSO

Este curso é destinado a usuários, especialistas de suporte e desenvolvedores de software que desejam aprender a utilizar o Linux, um ambiente computacional moderno, ágil e com um sistema operacional extremamente estável e versátil. O curso destina-se também aos administradores de sistemas Windows e aos profissionais que desejam iniciar os estudos para a certificação LPIC1, do Linux Professional Institute.

Este livro inclui os roteiros das atividades práticas e o conteúdo dos slides apresentados em sala de aula, apoiando profissionais na disseminação deste conhecimento em suas organizações ou localidades de origem.

ISBN 978-85-63630-19-3