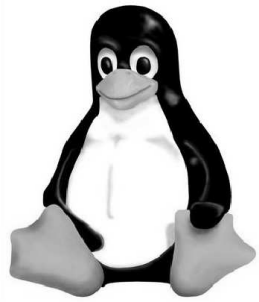
Conteúdo

Comandos

Prefácio



Mesmo com um significativo aumento do uso do Linux em computadores domésticos, e de sua vasta divulgação, é comum perceber o quanto as pessoas se intimidam ao se deparar com este famigerado Sistema Operacional. A verdade é que toda esta fama de "mau" deriva de tempos em que sua interface mantinha-se a "quilômetros" de distância do habitual sistema difundido em nossos computadores. Hoje, o Ubuntu por exemplo, uma das distribuições Linux, apresenta uma interface gráfica com suas funcionalidades muito semelhantes às de sistemas operacionais proprietários, tornando a "migração" do conhecimento intuitiva. Deixando as "aparências" de lado...

Linux é *Software Livre*. Esta poderia ser uma boa razão para a sua preferência a outros Sistemas Operacionais. Deixamos esta conclusão a cargo do leitor e vamos adiante. Em plena "Era Digital", não podemos deixar de falar em segurança, no que se refere a sistemas de computadores, uma das característica primárias visadas no desenvolvimento do Linux e seus aplicativos. Seja pelo resultado da comparação de prós e contras, ou até por necessidades profissionais, observamos cada vez mais usuários motivados a utilizá-lo.

Para que as diferenças relevantes entre o modo de uso dos Sistemas Operacionais não se tornassem um obstáculo acentuado, barrando a adoção do Linux como ferramenta cotidiana, seguindo a filosofia de Software Livre do Departamento de Informática da UFPR, os alunos do PET do curso de Ciência da Computação, elaboraram uma apostila básica sobre o assunto. Mais tarde, os alunos do PET passaram a oferecer curos semestrais, abertos à comunidade com este mesmo propósito.

Sobre a Apostila

Esta apostila foi escrita pelos integrantes do PET Computação da UFPR e é uma reformulação do mateiral escrito anteriormente. Abrange os conceitos fundamentais de organização do Linux e suas aplicações, voltadas ao *modo texto*. Além de uma referência aos interessados em adquirir conhecimento sobre este Sistema Operacional, esta nova abordagem segue a estrutura das aulas apresentadas, tradicionalmente pelo PET Computação na primeira semana de aulas, aos calouros do curso de Ciência da Computação, e este ano também, aos calouros do novo curso da UFPR, Informática Biomédica.

Divisão do Conteúdo

A seguir estão listados os pontos principais de cada capítulo. Ao final dos capítulos 3, 4 e 5 são propostos exercícios; alguns deles desafios, pois exigem informações sobre comandos que não serão apresentadas.

Cápiulo 1 ***Introdução***
: Apresenta conceitos básicos e um panorama sobre o Linux.

Cápiulo 2 ***Documentação***
: Descreve algumas maneiras de obter informações sobre comandos.

Cápiulo 3 ***Linux em Sua Essência***
: Apresenta a estruturação dos arquivos dentro do sistema e o espaço destinado aos usuários; modos de permissões, e alguns comandos para a administração do sistema.

Cápiulo 4 ***Mexendo Um pouco Mais***
: Tratamento da entrada e saída padrão de dados, redirecionamento; Gerência de processos.

Cápiulo 5 ***Terminal do Linux, Uma Mão na Roda***
: São descritos diversos comandos úteis para a manipulação de informações de arquivos contendo texto, os chamados *filtros*; trata da busca de arquivos e tabém da transferência de dados remotamente.

Apêndices
: Notas sobre a instalação de uma distribuição Linux e sobre a instalação de programas; breve descrição de alguns editores de texto “clássicos” do Linux.

Sugestões?

Esta apostila faz parte de um projeto do PET Computação que visa o constante aperfeiçoamento deste conteúdo, portanto críticas e sugestões são bem-vindas! Nosso e-mail? **pet@inf.ufpr.br**.

Curitiba, 6 de março de 2012

Capítulo 1

Introdução

1.1 Sistema Operacional

Um sistema operacional é um programa, ou um conjunto de programas, cuja função é gerenciar os recursos de um computador. Suas tarefas são: definir qual programa recebe atenção do processador, gerenciar memória, criar e gerenciar um sistema de arquivos, gerenciar dispositivos, além de fornecer uma interface entre o computador e o usuário. É o principal programa que a máquina executa desde o momento em que é ligada e mantêm-se em execução até que o computador seja desligado. O sistema operacional reveza sua execução com a de outros programas, como se estivesse vigiando, controlando e orquestrando todo o processo computacional.

A sigla usual para designar esta classe de programas é SO (em português) ou OS (do inglês Operating System). Ele pode ser dividido em duas grandes partes que cumprem funções complementares. São elas:

- O Núcleo, também chamado de Cerne, em português, ou Kernel, em inglês, que desempenha as funções vitais do sistema, gerenciamento, controle, monitoramento. Alguns autores consideram o Kernel sozinho como sendo, de fato, o SO;
- A concha, ou casca, mais comumente referenciada pelo seu nome original em inglês, Shell, que cumpre as funções de interface com o usuário, é o meio pelo qual o humano interage com o sistema. Pode apresentar-se em seu formato cru, via linha de comando, diretamente pelas entrada e saída padrão (teclado e monitor, respectivamente, no caso do PC), ou em roupagens mais amigáveis, tanto em modo texto quanto gráfico, com suporte a gerenciamento de janelas, e dispositivos de entrada e saída auxiliáres (mouse, dispositivos de áudio, leitores de linha braile, dentre outros.)

Segundo alguns autores, existem dois modos distintos de conceituar um sistema operacional, no caso, a função do Kernel:

- Pela perspectiva do usuário ou programador, visão top-down: é uma abstração do hardware, fazendo o papel de intermediário entre o aplicativo (programa) e os componentes físicos do computador (hardware);

- Numa visão bottom-up: é um gerenciador de recursos, i.e., controla quando e quais aplicações (processos) podem ser executadas. Assim como quando e quais recursos (memória, disco, periféricos) podem ser utilizados.

1.2 O Linux

Linux é o termo geralmente usado para designar qualquer sistema operacional que utilize o núcleo Linux. Este núcleo foi desenvolvido pelo finlandês Linus Torvalds, inspirado no sistema Minix. O seu código fonte está disponível sob licença GPL para qualquer pessoa que utilizar, estudar, modificar e distribuir de acordo com os termos da licença, i.e., simplificadamente, você pode alterar qualquer parte do Linux, construir qualquer sistema baseado nele e até comercialiazá-lo, mas você não pode fechá-lo, publica-lo sob uma licença proprietária e não permitir que outros usuários o modifiquem.

1.3 O GNU

GNU, sigla recursiva, que significa GNU Not Unix, é um projeto que começou em 1984 com o objetivo de desenvolver um sistema operacional completo, em software livre, mantido pela comunidade mundial de programadores, compatível com o padrão Unix. O Linux em si é somente um Kernel. Linus Torvalds, na mesma época que escrevia seu código-fonte, começou a usar programas da GNU para fazer seu sistema. Gostando da idéia, resolveu deixar seu kernel dentro da mesma licença e padrão. O kernel não pode ser usado por humanos diretamente. É necessária a existência de um Shell e, obviamente, dos programas, que executarão as funções desejadas pelo usuário.

O Kernel Linux, juntamente com aplicativos do projeto GNU, é agrupado, customizado, publicado, atualizado, mantido e distribuído para os usuários finais por várias empresas e organizações para os mais diversos fins. O resultado são as famosas distribuições GNU/Linux, dentre elas: Slakware, Gentoo, Debian, Mandriva, Red Hat e Ubuntu.

1.4 História do Linux

Na década de 1960 o Massachusets Institute of Technology (MIT), a General Eletric (GE), os laboratórios Bell (Bell Labs) e a American Telephone na Telegraph (AT&T) começaram o projeto de um sistema operacional de tempo compartilhado, onde vários usuários pudessem acessar os recursos de um único computador, sendo assim, o sistema mais arrojado da época. Seu nome era Multics, que em 1969, já rodava em um computador GE645. Os recursos computacionais disponíveis na época, no entanto, revelaram-se insuficientes para as pretensões do projeto e neste mesmo ano a Bell Labs se retirou do projeto.

Apesar disto, um de seus pesquisadores, Ken Thompson, continuou seus estudos no sistema com a intenção de criar algo menor mas que conservasse as suas idéias básicas. Usando um ocioso computador PDP-7, começou a escrever o Unics usando linguagem de montagem (assembly). Mais tarde, Brian Kernighan, também pesquisador da Bell Labs, rebatizou o novo sistema de Unix.

Um marco importante foi estabelecido em 1973 quando Dennis Ritchie, outro pesquisador da Bell, e Ken Thompson reescreveram o Unix, usando a linguagem de alto nível C, para um computador PDP-11. A linguagem C havia sido desenvolvida por Ritchie para substituir e superar as limitações da linguagem B, desenvolvida por Thompson. O seu uso é considerado uma das principais razões para a rápida difusão do Unix.

Entre 1977 e 1981, a AT&T alterou o Unix e lançou o System III. Em 1983, após mais uma série de modificações, foi lançado o conhecido Unix System IV, que passou a ser vendido. Até hoje esse sistema é usado no mercado como o padrão internacional do Unix. É comercializado por empresas como IBM, HP, Sun, etc. O Unix é um sistema operacional caro e é usado em computadores poderosos por diversas multinacionais.

Em 1987, Andrew Stuart Tanenbaum, escreveu um clone do Unix, chamado Minix, para o IBM PC com o intuito de ensinar a estudantes de ciência da computação como funciona um sistema operacional. O Minix foi escrito desde o começo, sem apreveitar nenhum código proprietário já existente do Unix. Consequentemente Tanenbaum escreveu um livro que descreve o sistema em detalhes e disponibizou o código em disquetes. Em três meses, na Usenet, um meio de comunicação semelhante a uma lista de discussão que data dos primórdios da Internet, milhares de leitores do mundo inteiro estavam discutindo e aprimorando o sitema. Um desses leitores era o estudante finlandês de ciência da computação Linus Torvalds da Universidade de Helsinki.

No fim dos anos 80, Linus teve contato com computadores compatíveis com o IBM PC, em 1990 começou a aprender C em seus estudos e em 1991 comprou um Intel 80386. Com 21 anos, ele tinha contato com o Sistema Unix da Universidade, o SunOS, atualmente Solaris, e desejava rodar a versão de Tannenbaum, Minix, no seu recém adquirido 80386. Entretanto, descontente com os recursos do Minix, especialmente em relação ao emulador de terminal que ele utilizaria para acessar remotamente o Unix da Universidade, começou a desenvolver o seu próprio emulador de terminal que não rodava sobre o Minix, mas diretamente no hardware do PC 386. Este projeto pessoal foi sendo modificado gradualmente e adquirindo características de um Sistema Operacional independente do Minix. Este foi o início do desenvolvimento do núcleo Freax (Free + Freak + X do Unix). Posteriormente batizado de Linux por Ari Lemmke, administrador do site ftp.funet.fi, que deu esse nome ao diretório FTP onde o núcleo Linux estava inicialmente disponível. O projeto foi lançado em 1991 em uma famosa mensagem para a Usenet em que Linus divulgou que estava disposto a disponibilizar o código-fonte e contar com a colaboração de outros programadores. Desde os primeiros dias, ele recebeu ajuda de hackers do Minix e hoje recebe contribuições de milhares de programadores dos mais diversos locais do mundo.

1.5 Uma Visão Geral

O linux é um sistema operacional livre, gratuito e possui todas as características presentes nos sistemas operacionais modernos: é um sistema multiplataforma, podendo ser operado em diversas arquiteturas; multiprocessado, possuindo suporte a computadores com mais de um processador; multitarefa, no qual vários programas podem ser executados ao mesmo tempo; e multiusuário, permitindo que mais de uma pessoa opere a máquina ao mesmo tempo.

O fato de ser baseado nos sistemas operacionais Unix - sistemas considerados

robustos, com mais de duas décadas de desenvolvimento - contribui muito para a sua confiabilidade e estabilidade. Se configurado corretamente, o sistema não irá travar. Este é outro ponto muito importante a favor do sistema: o fato de poder ser configurado e adaptado segundo a necessidade de cada um.

É possível instalar o Linux juntamente com outro sistema operacional. Por exemplo, pode-se instalar Linux em uma máquina que já contenha Windows e utilizar os dois sistemas sem que nenhum dado seja perdido. O Linux consegue acessar os arquivos usados no windows. Além disso, é possível utilizá-lo através de um CD sem que seja necessária a sua instalação.

O Linux, assim como todos os sistemas operacionais, utilizava inicalmente apenas interfaces de modo texto. A diferença é que no Linux a interface de modo texto evoluiu junto com o restante do sistema e se integrou de uma forma bastante consistente com os aplicativos gráficos. Quanto mais você aprende, mais tempo você acaba passando no terminal; não por masoquismo, mas porque ele é realmente mais prático para fazer muitas coisas.

O grande atrativo do terminal é que, com exceção de alguns poucos aplicativos específicos, os comandos são sempre os mesmos. Isso faz com que ele seja um porto seguro, com o qual você pode contar, sem importar em qual distribuição do Linux você está. O terminal é também a forma mais natural de "conversar"com o sistema sempre que você precisa de qualquer coisa além do arroz com feijão. Através dele é possível realizar qualquer operação.

Apesar disso, a utilização do terminal não é intuitiva. É preciso um conhecimento prévio para conseguir realizar as tarefas mais simples, o que não é atrativo para um leigo em computação. Isto levou a criação de uma mística de que o Linux é um sistema operacional muito complexo. O que muita gente ainda não sabe é que o Linux já dispõe de uma quantidade de ferramentas muito mais abrangente que a quantidade de ferramentas para Windows.

Nas distribuições mais atuais existem programas capazes de emular interfaces gráficas, proporcionando ao usuário um ambiente mais amigável, com janelas, botões e menus. Ao contrário do Windows, no Linux podemos escolher a interface gráfica que mais nos agrada em termos de beleza e facilidade de uso.

Capítulo 2

Documentação

Qualquer instalação do sistema Linux vem, por padrão, com diversos aplicativos já instalados, além de existirem milhares e milhares de pacotes disponíveis para instalação. Na hora de utilizar todas essas ferramentas sempre surgirão dúvidas a respeito da funcionalidade de certo aplicativo ou de seus parâmetros e opções.

Para sanar essas dúvidas e servir como guia de referência para todas as ferramentas e aplicativos instalados o sistema oferece uma documentação detalhada: as *man pages*.

2.1 Páginas de manual

Cada ferramenta tem sua própria página de manual, e todas seguem o mesmo padrão, com a mesma formatação e, em geral, as mesmas seções para tornar o uso dos manuais mais simples. Todas as páginas de manual seguem mais ou menos a seguinte estrutura:

NOME
: Nome do comando.

SINTAXE
: Exemplo da sintaxe genérica de uso do comando. Nesta seção há uma notação especial de representação:

  `negrito`
  : indica que deve-se digitar extamente como está escrito;

  `sublinhado`
  : deve ser substituído pelo conteúdo adequado;

  `...`
  : indica a possibilidade de repetição do conteúdo precedente;

  `[ ]`
  : todo o conteúdo entre colchetes é opcional;

  `|`
  : o contéudo delimitado pela barra vertical não pode ser usado em conjunto.

DESCRIÇÃO
: Breve descrição de sua funcionalidade.

OPÇÕES

Mostra todas as opções disponíveis e suas respectivas funções.

Para se ter acesso as páginas de manual, é preciso utilizar o comando **man** (comando 2.1, p. 12).

A mesma estrutura sintática é seguida pelas páginas de manual na hora de apresentar a estrutura de um comando. Além disso, as páginas de manual tem uma notação própria para indicar se um parâmetro ou opção é opcional e destacar texto a ser substituído: **opções e parâmetros aparecem entre colchetes quando opcionais. Texto a ser substituído aparece sublinhado.**

As páginas de manual ajudam muito o usuário Linux a conhecer mais sobre um certo comando e encontrar uma referência rápida a respeito de suas opções e parâmetros.

Além das páginas de manual existe uma outra fonte de informação disponível no sistema, contendo um material mais extenso e detalhado, com textos explicativos. Para acessar esse conteúdo usamos o comando **info**, que não está documentado nesta apostila. Tente **man** info.

2.2 /usr/share/doc

Uma outra alternativa na busca por informações sobre aplicativos é consultar o material encontrado no diretório /usr/share/doc/, que contém os arquivos *LEIAME* e informações sobre os autores de diversos programas instalados no sistema; além disso, é comum encontrar material no formato html com a documentação completa desses programas.

Comando 2.1 **man** Acessa páginas de manual.

NOME

man

SINTAXE

man [seção] comando
man -k palavra_chave

DESCRIÇÃO

O **man** acessa a página de manual do comando. É possível especificar a seção onde será buscada a página de manual. Os comandos estão organizados em seções cujos nomes são números de 1 a 9 e cada seção contém conjuntos de comandos com determinadas características. Mais detalhes **man** man.

O comando **man** também busca páginas de manual que contenham a palavra_chave em sua seção de descrição.

NOTAS

Para sair da página de manual digite [q]. Outras funcionalidades de navegação estão disponíveis, e para conhecê-las veja as páginas de manual do comando **less**.

Capítulo 3

Linux em Sua Essência

Vimos no capítulo anterior nosso primeiro comando, o **man**. Da mesma forma que este, todos os outros comandos apresentados são executados em *modo texto*, por um aplicativo chamado *terminal*, que será nossa forma de comunicação com o sistema Linux. O *modo texto* é o ambiente clássico de interface entre o usuário e sistemas baseados em UNIX, conhecido como *shell*, constituído por um interpretador de comandos textuais. Muitas distribuições Linux possuem interface gráfica, mas o bom e velho *shell* é superior em número de funcionalidades, possibilitando ao usuário, após um breve período de adaptação, trabalhar de forma mais eficiente em diversas tarefas.

Mas antes de abrirmos um terminal, é fundamental que estejam bem entendidos alguns conhecimentos básicos sobre Linux. Devemos ter em mente a estrutura de organização dos arquivos no sistema e onde está o nosso espaço de trabalho dentro desta organização. Ainda, quais as restrições de acesso a este espaço e como é feita sua interação com os demais usuários. Na sequência do capítulo, serão explicados alguns conceitos básicos e tarefas simples, pois antes de conhecer outras funcionalidades, o usuário deve saber como manter seus arquivos e se "localizar" no sistema.

3.1 Quem Está no Controle?

O Linux é um sistema multiusuário, ou seja, permite o acesso de várias pessoas ao mesmo tempo, tornando-o adequado à tarefa de servidor. Cada usuário possui um nome, ou *login*, e um número UID (*user identificator*), que o identifica no sistema. Para o gerenciamento desses usuários, existem estruturas que permitem dividi-los em grupos e também hierarquizá-los de acordo com restrições de acesso. Cada usuário pertence a no mínimo um grupo, que é composto por um nome e um GID (*group identificator*), número identificador.

O usuário *root*, único no sistema, pertence ao grupo root e ocupa o posto mais alto na hierarquia de permissões. Ele possui permissão para fazer qualquer coisa: acessar dados, excluí-los ou até mesmo apagar todo o sistema! Abaixo dele há os *sudoers*. Os sudoers são os usuários que podem utilizar o comando **sudo**, que permite executar tarefas como super usuário (root). Para que um usuário passe a ser sudoer, ele deve ter seu login ou um dos seus grupos listado no arquivo `/etc/sudoers` pelo root, seguido de quais comandos podem ser executados como

super usuário. Por fim, temos os usuário comuns, que não possuem previlégios.

Exercício Resolvido 3.1 *O administrador de um sistema precisa identificar e alterar a senha de todos usuários de um certo grupo. Quais linhas de comandos são mais adequadas para a ação?*

Comando 3.1 `whoami` Identifica o usuário logado.

NOME

whoami

SINTAXE

`whoami`

DESCRIÇÃO

O comando é utilizado para o usuário identificar-se na máquina. Normalmente, enta informação é apresentada no *prompt*[a]

[a]Veja sobre sobre *prompt* na seção [**?**]

Comando 3.2 `hostname` Mostra o nome da máquina.

NOME

hostname

SINTAXE

`hostname` `[-i|novo_nome]`

DESCRIÇÃO

Mostra o nome da máquina na rede ou altera seu nome para `novo_nome`.

OPCOES

`-i` mostra o endereço de IP da máquina.

Comando 3.3 `passwd` Altera senha de usuários.

NOME

passwd

SINTAXE

`passwd` `[-d|e] usuário]`

DESCRIÇÃO

`passwd` altera a senha do usuário.

OPCOES

`-d` remove a senha do usuário, perpitindo o livre acesso à sua conta;

`-e` expira a senha do usuário, forçando-o a trocar de senha na próxima vez que se logar.

Comando 3.4 `who` Mostra os usuários que estão logados.

NOME

who

SINTAXE

`who`

DESCRIÇÃO

Mostra quais usuário estão logados na máquina.

Comando 3.5 `finger` Lista informações sobre os usuários do sistema.

NOME

finger

SINTAXE

`finger` [usuário] ...

DESCRIÇÃO

O comando mostra informações sobre o usuário, tais como seu diretório pessoal, nome, data e hora do último login e tempo que está logado.

Comando 3.6 `su` Troca de usuário no terminal.

NOME

su

SINTAXE

`su` [usuário]

DESCRIÇÃO

O comando **su** troca de um usuário para outro no terminal corrente. Se o usuário não for especificado, será trocado para o root do sistema.

NOTAS

No sistema operacional Ubuntu, inicialmente não está definida uma senha do root. Para defini-la digite **sudo** `bash` e em seguida entre com a sua senha (caso você tenha feito a instalação do sistema). Em seguida use o comando **passwd** para definir a senha de root.

Comando 3.7 `sudo` Executa comandos com previlégios de root.

NOME

sudo

SINTAXE

`sudo` comando

DESCRIÇÃO

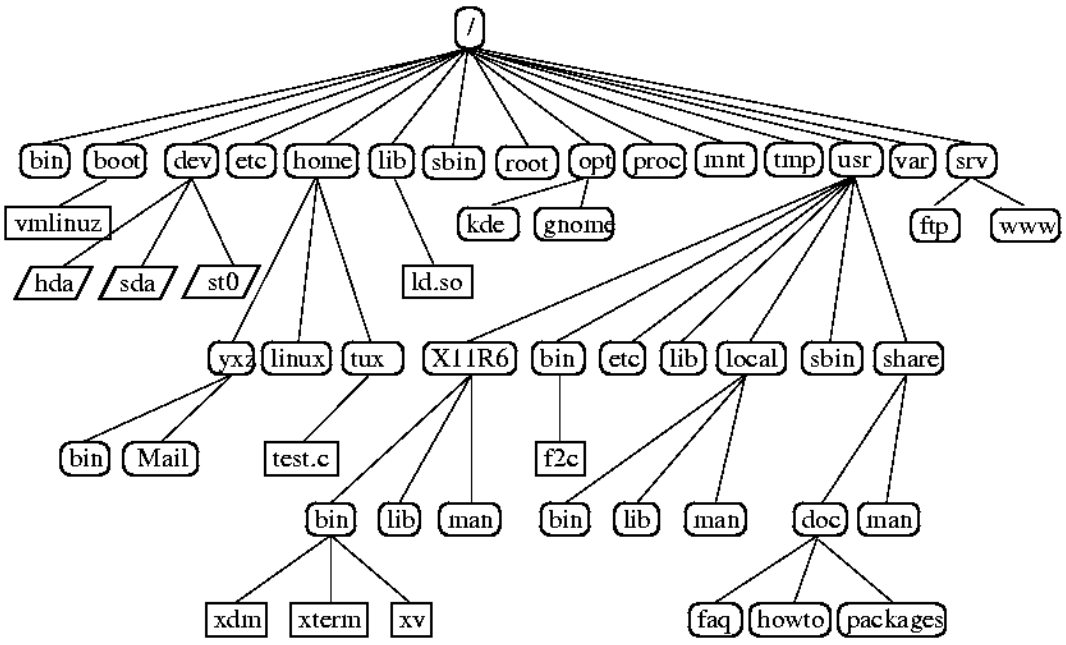
Executa comandos como administrador do sistema. Vários usuários podem ter este previlégio no sistema.

3.2 Arquivos e Pastas

Cada usuário do sistema possui uma pasta pessoal. Esta pasta, chamada *home*, é o local destinado ao armazenamento de todos os arquivos pessoais, com acesso restrito, se assim desejado, ao seu respectivo dono. Na linha de comando, a home pode ser representada com o metacaractere `~`, e a home de outros usuários, como a de Pedro, por exemplo, `~pedro`. Veremos abaixo onde se localizam estas pastas, e toda a estrutura de organização do sistema.

3.2.1 Estrutura Básica e Endereçamento

Os arquivos e diretórios encontram-se organizados em uma estrutura hierárquica na forma de árvore. Nessa estrutura, cada diretório está contido em outro e pode, por sua vez, conter vários diretórios e arquivos. Existe apenas um diretório que não está contido em nenhum, chamado raiz e representado por `/`. O diretório `/` é, como o nome diz, a raiz da árvore de diretórios: todos os outros arquivos e diretórios estão contidos em `/`.



Há duas formas de se fazer referência a diretórios e arquivos. Na primeira forma de referência usa-se o endereço relativo ao diretório corrente, diretório em que se está trabalhando.

```
$ whoami
chapolin
$ pwd
/home/pet/chapolin
$ cd Filmes/McGyver
$ pwd
/home/pet/chapolin/Filmes/McGyver
```

Outra forma é escrevendo o endereço completo, desde a raiz.

```
$ pwd
/home/pet/chapolin
$ cd /home/pet/Filmes/McGyver
$ pwd
/home/pet/chapolin/Filmes/McGyver
```

Note que, com esta forma de endereçamento, é possível referenciar facilmente quaisquer arquivos ou pastas, sem a necessidade de acessar o diretório em que se encontram tais objetos.

Por padrão, todos os diretórios contêm os diretórios "`.`" e "`..`". O caractere `.` representa o diretório corrente, ou seja, o próprio diretório em que se encontra. Já o `..` representa o pai do diretório corrente.

```
$ whoami
tirulipa
$ pwd
/home
$ cd ~
$ pwd
/home/pet/tirulipa
$ cd ..
$ pwd
/home/pet
```

Segue uma pequena apresentação dos principais diretórios do sistema.

`/boot` Encontramos o kernel e outros arquivos de inicialização;

`/bin` estão os executáveis mais simples;

`/usr` a maior parte dos executáveis instalados;

`/etc` guarda os arquivos de configuração;

`/dev` contém *links* para os periféricos;.

`/mnt` é o diretório padrão de montagem de periféricos de armazenamento;

`/home` encontram-se os diretórios pessoais de cada usuário.

3.2.2 Informações Sobre Arquivos

O formato dos arquivos não é decidido pela sua extensão, como no Windows, mas pela sua estrutura interna. Logo, a extensão de um arquivo serve apenas para fins de organização e para facilitar futuras buscas. Outro ponto importante é que em nomes de arquivos ou pastas há distinção entre letras maiúsculas e minúsculas. Um arquivo chamado `RincaoAlegre.jpg` não será encontrado se referenciado como `rincaoAlegre.jpg`, por exemplo.

Cada arquivo guarda três informações de tempo: *access*, *modify* e *change*. *Access* é atualizado quando se acessa um arquivo. *Modify* é quando o conteúdo do arquivo é modificado. *Change* é modificado quando *modify* se atualiza ou quando ocorre uma alteração das propriedades externas do arquivo, como permissões, ou quando ele é movido para outro diretório.

3.2.3 *Links*

Um *link* é uma forma de se referenciar arquivos e pastas. Há dois tipos de *links*: os *links* simbólicos e os *hard links*.

Um *link* simbólico é um arquivo cujo conteúdo é o endereço de um alvo, que pode ser um diretório ou de outro arquivo. Através do *link* simbólico, podemos ler e modificar o alvo. Se o alvo for apagado ou trocado de lugar, o *link* fica órfão. O alvo pode estar em outra partição.

Um *hard link* é uma referência direta à posição no disco apontada pelo alvo. Este tipo de *link* só pode ser usado em arquivos. O *hard link* tem as mesmas propriedades do arquivo original. Se o alvo for apagado, o *link* continua como estava. Para apagar um arquivo, é necessário apagar todos os seus *hard links*.

Exercício Resolvido 3.2 *Você está no diretório* `/pasta_exercicio/Diversos`. *Como copiar todos os arquivos deste diretório diretamente para sua home?*

```
cp * ~
```

Primeiramente usamos o comando **`cp`**, *que copia os arquivos. A opção* * *irá incluir todos os arquivo do diretório atual, e o* ~ *indica o destino desse arquivo. Observe que nesse caso, pastas contidas no diretório corrente não serão copiadas, a menos que a opção* **`-r`** *seja utilizada.*

Exercício Resolvido 3.3 *O diretório* `/pasta_exercicios` *está na sua home. Como excluir todos os arquivos e diretórios contidos nesta pasta?*

```
$ cd ~ $ rm -r pasta_exercicios/*
```

Juntamente com o comando **`rm`**, *a opção -r irá excluir o diretório e todos diretórios abaixo dele.*

Comando 3.8 `pwd` Mostra o endereço completo do diretório atual.

NOME

pwd

SINTAXE

`pwd`

DESCRIÇÃO

Mostra o endereço completo do diretório atual.

Comando 3.9 `cd` Muda de diretório.

NOME

cd

SINTAXE

`cd` [diretório]

DESCRIÇÃO

O comando **`cd`** muda para o diretório. Se nenhum parâmetro for passado, muda para o diretório `home` do usuário.

Comando 3.10 `ls` Lista o conteúdo de diretórios e informações sobre arquivos.

NOME

ls

SINTAXE

`ls` [**`-lahd`**] [arquivo] ...

DESCRIÇÃO

O comando **`ls`** lista o conteúdo do diretório arquivo, ou suas informações, se arquivo não for um diretório. Quando usado sem parâmetros, lista o conteúdo do diretório atual.

OPÇÕES

`-l` Exibe informações mais detalhadas sobre cada arquivo/diretório. As informações exibidas são: permissões, grupo e dono, tamanho, tempo da última modificação e seu nome.

`-a` Exibe arquivos ocultos.

`-h` A opção **`-l`** mostra o tamanho dos arquivos em bytes. Esta opção apresenta o tamanho dos arquivos com uma unidade apropriada em função destes tamanhos, tornando-os legíveis.

`-d` Não lista o conteúdo de diretórios, mas sim suas informações.

Comando 3.11 `mkdir` Cria diretórios.

NOME

mkdir

SINTAXE

`mkdir` diretório ...
`mkdir -p` diretórios/subdiretório ...

DESCRIÇÃO

Cria diretórios.

OPÇÕES

`-p` Ao criar o subdiretório, cria também todos os diretórios inexistentes.

Comando 3.12 `rmdir` Remove diretórios vazios.

NOME

rmdir

SINTAXE

`rmdir` `diretório ...`

DESCRIÇÃO

Remove diretórios vazios.

Comando 3.13 `rm` Remove arquivos e diretórios.

NOME

rm

SINTAXE

`rm` `[-rif] arquivo ...`

DESCRIÇÃO

Remove arquivos e diretórios permanentemente.

OPÇÕES

`-r` Remove diretórios e todo seu conteúdo recursivamente.

`-i` Pede por confirmação antes de remover cada arquivo.

`-f` Força a exclusão dos arquivos: não pede confirmação nem retorna erros para arquivos inexistentes.

Comando 3.14 `mv` Move arquivos e diretórios.

NOME

mv

SINTAXE

`mv` `[-i] arquivo ... destino`
`mv` `[-i] arquivo destino/novo_nome`

DESCRIÇÃO

O comando **`mv`** move o `arquivo` para o diretório `destino`. Se mais de um arquivo for passado como parâmetro, `destino` deve ser um diretório. Caso contrário é possível renomear o `arquivo` com o `novo_nome` ao movê-lo para `destino`.

OPÇÕES

`-i` Pede confirmação antes de sobrescrever qualquer arquivo.

Comando 3.15 `cp` Copia arquivos e diretórios.

NOME

cp

SINTAXE

`cp` [**`-ri`**] `arquivo ... destino`
`cp` [**`-ri`**] `arquivo destino/novo_nome`

DESCRIÇÃO

O comando **`cp`** copia o `arquivo` para o diretório `destino`. Se mais de um arquivo for passado como parâmetro, `destino` deve ser um diretório. Caso contrário é possível nomear a cópia do `arquivo` como `novo_nome`.

OPÇÕES

`-r` Copiar diretórios, com todo seu conteúdo, recursivamente.

`-i` Pede confirmação antes de sobrescrever arquivos existentes.

Comando 3.16 `file` Mostra o formato do arquivo.

NOME

file

SINTAXE

`file` `arquivo ...`

DESCRIÇÃO

Mostra o formato do arquivo.

Comando 3.17 `stat` Mostra informações sobre arquivos.

NOME

stat

SINTAXE

`stat` `arquivo ...`

DESCRIÇÃO

Este comando mostra o formato do arquivo, o tamanho, a posição no HD, as permissões, o login e o grupo do dono e os tempos *access*, *modify* e *change*.

OPÇÕES

`-f, --file-system`
Mostra *status* de um sistema de arquivos ao invés de um arquivo.]

Comando 3.18 `touch` Atualiza as datas de um arquivo.

NOME

touch

SINTAXE

`touch` `arquivo ...`

DESCRIÇÃO

Atualiza os tempos *modify*, *change* e *access* do `arquivo`. Se o `arquivo` não existir, será criado.

Comando 3.19 `ln` Cria *links* de arquivos.

NOME

ln

SINTAXE

`ln` `[-s] arquivo link`

DESCRIÇÃO

Cria um link do `arquivo`. Se for usado sem opções, cria um *hard link*.

OPÇÕES

`-s` Cria um *link* simbólico.

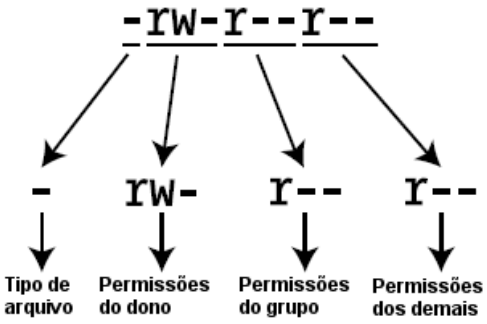
3.3 Permissões

Para entender sobre permissões, será necessária a compreensão de outros dois conceitos vistos anteriormente sobre arquivos e usuários. Permissões são um dos aspectos mais importantes do Linux (na verdade, de todos os sistemas baseados em Unix), pois elas são usadas para várias finalidades, mas servem principalmente para proteger o sistema e os arquivos dos usuários.

Somente o root tem ações irrestritas no sistema. Naturalmente, a especificação do que cada usuário ou grupo do sistema pode fazer é determinada através de permissões. Ao listar as permissões de um arquivo ou pasta, estas serão mostradas no seguinte formato:

```
$ ls -l
drwxr-xr-x 2 chapolin pet-user 4096 2011-02-25 16:22 Comandos
-rw-r--r-- 1 chapolin pet-user  392 2011-02-25 14:37 Makefile
```

O primeiro item que aparece em cada linha, `drwxr-x-r-x` e `-rw-r--r--` acima, é a forma usada para mostrar as permissões do diretório `Comandos` e do arquivo `Makefile`.Um ponto interessante é que o Linux trata **todos os diretórios como arquivos**[1], portanto, as permissões se aplicam de igual forma para ambos. Tais permissões podem ser divididas em quatro partes para indicar: tipo, proprietário, grupo e outras permissões.



O primeiro caractere da *string* indica o tipo de arquivo: se for `d` representa um diretório, caso seja `-` equivale a um arquivo. Entretanto, outros caracteres podem aparecer para indicar outros tipos de arquivos, conforme abaixo.

d Diretório;

b arquivo de bloco;

c arquivo especial de caractere;

p canal;

s socket;

- arquivo “comum”.

[1] Portanto, os comandos da apostila que se aplicam a “arquivos”, certamente valem para diretórios, a menos que seja explicitado no texto.

No restante da *string* (conjunto de caracteres) estão as permissões propriamente ditas. São 9 caracteres divididos em grupos de 3, que representam as permissões de determinados perfis de usuários. Agora que já sabemos o significado das divisões da *string*, vamos entender o que as letras **r**, **w**, **x** e o caractere **-** representam. No caso de arquivos:

r Significa permissão de leitura (*read*);

w Significa permissão de gravação (*write*);

x Significa permissão de execução (*execution*);

- Significa permissão desabilitada.

Ja para diretórios:

r Significa permissão para listar o conteúdo (*read*);

w Significa permissão para criar e excluir arquivos e diretórios (*write*);

x Significa permissão para entrar no diretório (*execution*);

- Significa permissão desabilitada.

Leitura permite ao usuário ler o conteúdo do arquivo mas não alterá-lo, ou listar o conteúdo se for uma pasta. Gravação permite que o usuário altere o arquivo ou o interior do diretório. Execução, como o nome diz, permite que o usuário execute o arquivo, no caso de ser executável e, no caso de pastas, acessá-la.

Estas permissões,**r**, **w** e **x**, não funcionam isoladamente, ou seja, de forma que o usuário tenha ou permissão de leitura ou de gravação ou de execução, elas funcionam em conjunto.

Existem duas maneiras de expressar as permissões: forma textual e forma octal.

3.3.1 Notação Textual

A notação textual é mais maleável, e pode ser escrita de algumas formas, como veremos abaixo.

```
$ chmod (ugoa)(+-=)(rwx) arquivos
```

O primeiro grupo de caracteres refere-se a quem a permissão será alterada.

u Permissão do usuário proprietário;

g Permissão do grupo;

o Permissão dos demais usuários;

a Permissão de todos os usuários, ou seja, proprietário, grupo e outros usuários.

O segundo grupo de caracteres refere-se ao tipo de alteração que será feita.

+ Adiciona uma permissão;

- Remove uma permissão;

= Define estado final da permissão de tipo de usuário;

O terceiro grupo de caracteres refere-se ao tipo de permissão, já comentado acima. Para melhor entendimento, veja o exemplo:

```
$ ls -l
-rw-rw-r-- 1 chapolin pet-user  392 2011-02-25 14:37 Makefile
$ chmod o-r Makefile
$ ls -l
-rw-rw---- 1 chapolin pet-user  392 2011-02-25 14:37 Makefile
$ chmod g=r Makefile
$ ls -l
-rw-r----- 1 chapolin pet-user  392 2011-02-25 14:37 Makefile
$ chmod og+w Makefile
$ ls -l
-rw-rw--w- 1 chapolin pet-user  392 2011-02-25 14:38 Makefile
$ chmod o=rx Makefile
$ ls -l
-rw-rw-r-x 1 chapolin pet-user  392 2011-02-25 14:38 Makefile
```

Para realizar trocar completas, devem ser utilizadas vírgulas para separar os usuários:

```
$ ls -l
-rw-rw-r-x 1 chapolin pet-user  392 2011-02-25 14:38 Makefile
$ chmod u=rw,g=r,o= Makefile
$ ls -l
-rw-r----- 1 chapolin pet-user  392 2011-02-25 14:38 Makefile
```

3.3.2 Notação Octal

Além da forma textual para descrever as permissões vista acima, existe também o modo numérico, chamada de forma octal. Basicamente deve se pensar no sistema numérico binário, substituindo-se os tipos de permissões por 0s ou 1s. Se determinada permissão é habilitada, atribui-se valor 1, caso contrário, atribui-se o valor 0. Assim, a string de permissões `r-xr-----` na forma numérica torna-se 101100000. Essa combinação de 1's e 0's é um número binário. Mas temos ainda que acrescentar a forma decimal (ou seja, representados por números de 0 a 9). Para isso, observe a tabela abaixo:

String	Binário	Decimal	Permissões
---	000	0	Nenhuma permissão
--x	001	1	Executar
-w-	010	2	Escrever
-wx	011	3	Escrever e Executar
r--	100	4	Ler
r-x	101	5	Ler e Executar
rw-	110	6	Ler e Escrever
rwx	111	7	Ler, Escrever e Executar

Para exemplificar, vamos utilizar a permissão `rw-`, cujo valor em binário é 110, que por sua vez, em decimal corresponde ao número 6. Então, em vez de usar `rw-` ou 110 para criar a permissão, simplesmente usa-se o número 6. Repare que, com o método numérico, usamos somente um dígito para representar uma permissão, em vez de três. Assim, a string de permissões `r--r--r--` pode ser representa por 444, pois `r--` em decimal é igual a quatro. Caso você desconheça o sistema numérico binário, existe outra maneira de pensar:

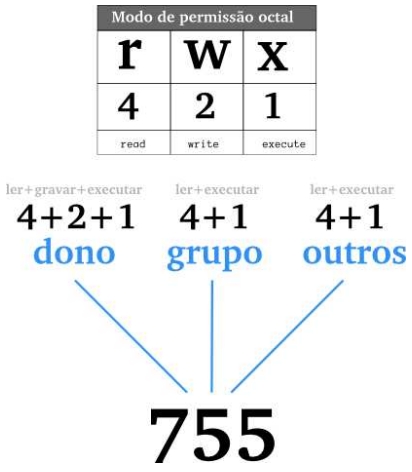
4 Permissão de leitura;

2 permissão de escrita;

1 permissão de execução;

0 indica sem permissões.

Agora é simples, é só somar as permissões. A permissão `rwxrw-r--` em octal, por exemplo, ficaria assim: 4 + 2 + 1 = 7 (permissão de `rwx`), 4 + 2 = 6 (permissão `rw`) 4 = (permissão `r`). Para que não fique dúvidas, veja mais um exemplo com a figura abaixo.



Exercício Resolvido 3.4 *Estando em sua home, como alterar as permissões de todos os diretórios e arquivos do diretório `/home/user/pasta_exercicios`?*

```
$ chmod -R 744 pasta_exercicios
```

A partir de sua home, com o comando `chmod` e a opção -R, é possível alterar as permissões de todos os arquivos do diretório `/pasta_exercicios`recursivamente, ou seja, alteraremos as permissões de todos os diretórios e arquivos existentes dentro do diretório `/pasta_exercicios`.

Comando 3.20 `chmod` Troca permissões de arquivos ou diretórios.

NOME

chmod

SINTAXE

`chmod` `[-fR] permissões arquivo ...`

DESCRIÇÃO

O comando chmod troca as permissões de um arquivo ou diretório. Para especificar as novas `permissões` pode ser utilizado as duas formas vistas anteriormente: o modo octal, em que a descrição da permissão será feita de forma completa,

```
$chmod 744 arquivo.txt
```

e o modo textual. Neste modo as permissões podem ser descritas parcialmente, apenas por grupos ou usuários, ou apenas por uma permissão específica, por exemplo, adicionar permissão de leitura ao grupo.

```
$chmod g+r arquivo.txt
```

OPÇÕES

`-f` Suprime mensagens de erro, ou seja, executa o comando sem mostrar erros na tela;

`-R` Troca a permissão dos arquivos e diretórios recursivamente.

Comando 3.21 `chown` Troca proprietário do arquivo.

NOME

chown

SINTAXE

`chown` `[-fR] [novo_proprietário] [:[novo_grupo]] arquivo ...`

DESCRIÇÃO

Este comando permite trocar o usuário e/ou o grupo a quem um determinado arquivo pertence. Este é um comando restrito apenas ao root ou outros administradores e pode ser utilizado da seguinte forma:

```
$chown novo_roprietário arquivo.txt
```

Troca apenas o proprietário do arquivo.

```
$chown novo_proprietário:novo_grupo arquivo.txt
```

Troca o proprietário e o grupo do arquivo.

```
$chown NovoProprietário: arquivo.txt
```

Troca o proprietário e o grupo do arquivo. Sendo que o novo grupo é o grupo do novo proprietário do arquivo.

```
$chown :NovoGrupo arquivo.txt
```

Troca o apenas o grupo do arquivo. Funciona da mesma maneira que o comando **`chgrp`**.

OPÇÕES

`-f` Suprime mensagens de erro, ou seja, executa o comando sem mostrar erros na tela;

`-R` Troca o proprietário e/ou o grupo dos arquivos e diretórios recursivamente.

NOTAS

O caractere `:` é equivalente ao caractere `.` neste comando, podendo ser utilizado em sua sintaxe.

Comando 3.22 `umask` Determina as permissões padrão para novos arquivos e diretórios criados.

NOME

umask

SINTAXE

umask permissões

DESCRIÇÃO

Quando o usuário cria um arquivo ou diretório, o sistema associa ao objeto criado um conjunto de permissões padrão de acesso. Este valor padrão é definido pelo comando **umask**. Este comando pode ser executado de duas maneiras: forma textual e forma octal. Na forma textual, este funciona como explicado nesta seção, exemplo:

```
$umask u=rwx,g=rx,o=
```

Já na forma octal existe um pequeno detalhe que faz uma grande diferença. Deve-se informar as permissões que **não** serão dadas, ao invés das permissões desejadas. Outra forma de se pensar é subtrair a permissão desejada da permissão máxima (777). Vamos refazer o exemplo acima, agora na forma octal:

```
$umask 027
```

A permissão desejada é 750, então subtrai-se 750 da permissão máxima (777), obtendo-se o valor do **umask** 027.

NOTAS

É muito importante saber que o comando **umask** funciona apenas no terminal executado, após seu encerremento o efeito do comando é perdido. Para alterar em todos os terminais é necessário alterar o arquivo de configuração de seu usuário[a].

[a] ver seção da nicoly

3.4 Exercícios

Para resolver esses desafios, utilize as man pages, juntamento com as informaçoes da apostila. Não esqueça de fazer o download do diretório de exercicios que acompanha a apostila.

Exercício 3.5 *Se um diretório tiver as permissões* `drwx--x---`*, e dentro deste diretório existe um arquivo, cujas permissões são* `-rwx--x---`*. Para quem o dono do arquivo quer disponibilizá-lo? Provavelmente, que tipo de arquivo é este?*

Exercício 3.6 *Como ficariam as permissões de um arquivo que está definido como somente leitura para o grupo do dono do arquivo e também para os outros usuários que não fazem parte deste grupo, e leitura e escrita para o dono do arquivo?*

Exercício 3.7 *Você fez o download de um filme, mas na pasta deste filme veio um arquivo deconhecido, que pode ou não ser do seu interesse. Como pode-se obter informações desse arquivo sem executá-lo? Quais as implicações da ação anterior na segurança do sistema?*

Exercício 3.8 *Você é o administrador de um sistema, e o problema é o seguinte: existe um arquivo importante na home de um usuário, você conhece o login deste usuário. A partir desta única informação, qual a sequência de comandos apropriada para copiar este arquivo para sua home, ou ao menos ter acesso a esse arquivo a partir de sua home?*

Capítulo 4

Mexendo um pouco Mais

Neste capítulo trataremos de um assunto que é muito importante quando se quer entender como linhas de comandos funcionam, que é a entrada e saída de dados. Basicamente, estas duas palavras são o essencial para o entendimento de como funciona o terminal. Normalmente uma entrada é fornecida, e um resultado é gerado. A partir do momento em que o usuário conseguir entender essa lógica básica, a utilização de comandos se tornará mais simples e sem mistérios.

Além disso, aqui também será explicado a fundo como o interpretador de comandos shell trabalha, a importância de suas variáveis de ambiente e dos seus arquivos de configuração, além de como saber analisá-los e configurá-los para melhor atender cada usuário. Essa personalização é de grande utilidade, pois gera uma boa economia de tempo para o usuário.

Outro ponto importante que será abordado diz respeito à administração de um sistema operacional. Será mostrado e exemplificado como aplicativos podem ser controlados através do terminal, e até mesmo por simples combinações de teclas. Além do citado acima, uma facilidade muito grande que o shell nos oferece é em respeito ao controle dos dados do sistema. Tarefas que envolvem unidades de armazenamento e controle de dados em disco se mostram práticas quando executadas via shell.

4.1 Nada se Cria Tudo se Transforma!

4.1.1 Entrada e saída de dados

A grande maioria dos comandos utilizados em terminal requer dados passados como parâmetros para sua execução. Estes dados são fornecidos por uma entidade chamada de entrada, que pode ser o teclado ou um arquivo.

A execução de um comando utiliza os dados fornecidos pela entrada para gerar um resultado, que é mostrado através de uma entidade chamada de saída. Esta saída pode ser a própria tela do terminal ou um arquivo também.

A escolha do tipo de entrada e saída depende do comando que está sendo utilizado. Alguns comandos nos permitem escolher estas informações através dos parâmetros. Outros já possuem entrada e saída definidas, as quais são chamadas de entrada e saída padrão.

No shell a entrada padrão é o teclado e a saída padrão é o próprio terminal. Em geral, quando um comando requer parâmetros de entrada e saída e nada for especificado a entrada e saída padrão serão usadas.

Vejamos um exemplo com o comando cat, que serve basicamente para imprimir o conteúdo do arquivo de entrada:

```
$cat arquivo.txt
```

O comando cat admite como parâmetro apenas a entrada, a saída será por padrão o terminal. Esta saída pode ser alterada por meio de redirecionamentos, que veremos mais adiante.

Caso um arquivo de entrada não seja fornecido, esse programa irá esperar dados serem digitados no próprio terminal, para depois serem impressos. Para interromper, deve-se utilizar Control+d [1]. Vejamos o exemplo a seguir:

```
$cat
Testando texto
Testando texto
Denovo
Denovo
^D
```

Como o **cat** não recebeu nenhum parâmetro ele leu os dados da entrada padrão e então os enviou para a saída padrão, por isso todo texto digitado é duplicado.

4.1.2 Redirecionamento destrutivo e não-destrutivo

Um artifício muito útil é a possibilidade de redirecionar entradas e saídas utilizando símbolos.

Os símbolos **>** e **>>** são usados para modificar a saída de um comando, enviando os dados para um arquivo, ao invés de utilizar a saída padrão. Observe este exemplo:

```
$ls
```

Usado deste modo, o resultado é uma lista de arquivos e pastas do diretório atual mostrada no próprio terminal das informações (saída padrão). Utilizando os símbolos, temos:

```
$ls > arquivos.txt
```

Esta linha de comando não imprimirá nada no terminal. Ao invés disso, irá criar o arquivo `arquivos.txt`, com as informações de saída do comando ls.

O símbolo **>** é chamado de redirecionador destrutivo, pois caso seja fornecido um arquivo já existente, ele apaga todas as informações do mesmo para inserir as novas.

Já o símbolo **>>** é chamado de não-destrutivo. Ele concatena as informações no final do arquivo, caso o mesmo já exista.

Quando estamos falando de dados de entrada, o símbolo **<** pode nos poupar muito tempo. Seu funcionamento é muito simples, ele redireciona os dados contidos em um arquivo para a entrada de um comando. Imagine que o comando é

[1] Quando um programa está lendo dados de um arquivo ele termina a entrada de dados quando lê um valor específico que representa o final do arquivo. Quando estamos entrando os dados via teclado, precisamos pressionar Control+D para simular a leitura deste valor.

executado e a cada vez que a entrada padrão é solicitada, a leitura é feita diretamente no arquivo. Veja o exemplo a seguir.

O comando sort recebe dados de entrada e os ordena, enviando para a saída padrão:

```
$sort
```

O resultado é o terminal esperando dados serem digitados, e só então a ordenação é realizada.

```
$sort < arquivo.txt
```

Neste caso o resultado será imediato. Os dados contidos no arquivo serão ordenados e impressos na saída padrão. Agora observe o seguinte exemplo:

```
$sort < arquivo1.txt > arquivo2.txt
```

Este comando não irá imprimir nada, apenas ordenar os dados do arquivo1 e jogá-los no arquivo2. Observe que é possível combinar os símbolos para obter resultados diferentes.

Analisando a sintaxe do comando sort, observamos que ele aceita como parâmetro um arquivo de entrada, então é possivel utilizar o comando:

```
$sort arquivo.txt
```

O resultado será o mesmo de `$sort < arquivo.txt`.

Quando estamos trabalhando com programas em C ou outras linguagens, podemos utilizar o redirecionador de entrada para fazermos testes ao alterarmos partes do código. Observe a seguinte situação:

Imagine que você fez um programa que recebe como entrada 100 números e imprima a média aritmética deles. Você o compilou e precisa testar seu funcionamento. Você o executa e deve digitar os 100 números, um a um. Imagine que ocorra um erro no 98^o número. O próximo passo é achar o erro e testar novamente. O problema é que em cada teste, cem números devem ser digitados. Cansativo!

É muito mais prático criar um arquivo com os cem números, e simplesmente redirecioná-lo como entrada de dados:

```
$./calcula_media < dados.txt
```

E neste caso utilizar um redirecionamento da entrada padrão é muito interessante por tornar a construção do programa em C ou qualquer linguagem muito mais simples. Criar um programa que lê os dados da entrada padrão é muito fácil, enquanto criar um programa que lê os dados de um arquivo exige o uso de diversas rotinas de manipulação de arquivos.

4.1.3 Pipe

E se quisermos redirecionar a saída de um comando, não para um arquivo, mas para a entrada de outro comando, seria possível? A resposta é sim!

Existe um símbolo muito poderoso no shell, que executa esta tarefa, chamado de Pipe.

Com ele podemos reduzir o número de instruções, ou seja, uma tarefa que seria realizada com várias linhas de comando, pode ser realizada em apenas uma. Veja

o seguinte exemplo: imagine que queremos guardar em um arquivo o nome de todos os arquivos e pastas do diretório corrente, mas em ordem alfabética. Uma tarefa simples, realizada com os seguintes comandos:

```
$ls > dados.txt
$sort dados.txt > ordenados.txt
```

A tarefa foi executada, porém tivemos que utilizar dois comandos e ainda um arquivo temporário.

Utilizando pipe a mesma tarefa é executada pela seguinte linha de comando:

```
$ls | sort >> ordenados.txt
```

Observe que neste caso não há arquivo temporário, apenas o destino final. O funcionamento deste símbolo é simples, vamos passo a passo na linha de comando anterior:

O comando **ls** por padrão imprime os dados no terminal, porém a presença do pipe faz com que esses dados sejam redirecionados para a entrada do próximo comando. O sort é executado sem a utilização de um parâmetro, pelo fato de que os dados já foram enviados do comando anterior. Nesta etapa, os dados são ordenados e redirecionados para o arquivo final.

Vale ressaltar que vários pipes podem ser utilizados, tornando possível a realização de tarefas complexas com apenas uma linha de comando.

4.1.4 Saída de erros

Além da entrada e saída padrão, existe uma entidade chamada saída de erros, que serve para a impressão de erros. A saída de erros padrão é a tela do terminal.

Alguns comandos podem emitir mensagens de erro, caso não funcionem como esperado e não retornem o resultado desejado. Por exemplo o comando ls:

```
$ls Teste
```

Esta linha de comando faz com que seja impresso todos os arquivos e pastas contidos na pasta Teste. Mas, e se a pasta Teste não existisse?

Neste caso, seria impresso na saída de erros a seguinte mensagem:

```
$ls: Teste: No such file or directory
```

Em alguns casos é interessante transportar as mensagens de erros que ocorrem no sistema para um arquivo, para serem analisadas posteriormente ou para que os erros não sejam impressos no terminal. Para isso, utilizamos o símbolo **2>** que funciona exatamente da mesma forma que o redirecionamento de saída de dados **>**:

```
$ls Teste 2> erros.txt
```

Exercício Resolvido 4.1 *Como concatenar conteúdo usando o comando tee?*

```
$ cat nomes.txt Frodo
Sam
Merry
Bilbo
$ whoami | tee -a nomes.txt
$ cat nomes.txt
Frodo
Sam
Merry
Bilbo
Pet
```

No diretório `/pasta_exercicios/Diversos`*, com a opção -a, podemos concatenar conteúdo ao conteúdo antigo de um arquivo passado como parâmetro ao comando tee.*

Exercício Resolvido 4.2 *Por que usar o comando xargs?*

```
$ ls
Ace_of_Spades.txt Be_my_Baby.txt Sucker.txt
$ cat > apagar.txt
Sucker.txt
^D
$ ls
Ace_of_Spades.txt apagar.txt Be_my_Baby.txt Sucker.txt
$ cat apagar.txt
Sucker.txt
$ cat apagar.txt | xargs rm
$ ls
Ace_of_Spades.txt apagar.txt Be_my_Baby.txt
```

Considerando o arquivo `/apagar_exercicios/Musicas/apagar.txt`*, com o comando* `cat` *vemos o conteúdo como sendo 'Sucker.txt'. Porém, se redirecionarmos esse resultado, usando o pipe, para o comando* `xargs`*, o conteúdo do arquivo* `/pasta_exercicios/Musicas/apagar.txt` *vira argumento do comando que segue o* `xargs`*. Ou seja, o comando executado será o 'rm Sucker.txt'. O comando* `xargs` *é muito interessante quando precisamos repassar pelo pipe o argumento de algum comando, como por exemplo o nome do arquivo que o comando* `rm` *deve apagar, o nome do usuário que o comando* `finger` *deve procurar ou o nome do arquivo que o comando* `touch` *deve criar. Vale lembrar que o pipe nem sempre precisa ser usado seguido do* `xargs`*. Existem vários comando que podem manipular a informação que está sendo repassada pelo pipe, ou seja, comandos que procuram uma linha específica desse conteúdo, que contam o número de linhas do arquivo, etc. Em resumo, usamos o* `xargs` *para modificar arquivos (criar , excluir, compactar), enquanto que para apenas modificar um conteúdo que será apenas "jogado" na saída padrão, sem que nenhuma modificação seja realmente feita no arquivo, podemos usar outros comandos sem o* **`xargs`** *(ver Seção 5.3).*

Comando 4.1 `cat` Envia dados para a saída padrão.

NOME

cat

SINTAXE

`cat [opcões] [entrada]`

DESCRIÇÃO

O comando cat imprime o arquivo na saída padrão. Caso mais de um arquivo seja fornecido, suas informações são concatenadas e então impressas. A utilização do comando sem o fornecimento de um arquivo imprime dados da entrada padrão.

OPÇÕES

`-b` Enumera linhas não em branco.

`-n` Enumera todas as linhas.

`-s` Suprime linhas em branco repetidas.

Comando 4.2 `more` Visualizar arquivos.

NOME

more

SINTAXE

`more [opcões] [entrada]`

DESCRIÇÃO

O comando more é utilizado para visualizar arquivos. Diferentemente do comando cat, o more não imprime diretamente todo o conteúdo, mas sim de partes em partes, de acordo com o tamanho da janela do terminal. Deste modo é possível a leitura de arquivos longos.
Com este comando, a ordem de visualização é de cima para baixo.

OPÇÕES

`-s` Suprime linhas em branco repetidas.

Comando 4.3 `less` Visualizar arquivos.

NOME

less

SINTAXE

`less` [opcões] [entrada]

DESCRIÇÃO

O comando realiza a tarefa básica do comando more porém com algumas funcionalidades a mais. Possui a capacidade de exibir o arquivo com opção de rolagem para trás e para frente. Como não necessita ler todo o arquivo de entrada antes de exibí-lo, consegue maior desempenho comparado a outros visualizadores e editores quando manipulam arquivos grandes.

OPÇÕES

`-N` Enumera todas as linhas

`-s` Suprime linhas em branco repetidas.

EM EXECUÇÃO

ESPACE
: próxima página

ENTER
: próxima linha

b
: topo

v
: editor de texto vi

q
: sair

h
: lista todos os comandos

Comando 4.4 `xargs` Passa dados da entrada padrão como argumentos.

NOME

xargs

SINTAXE

`xargs` [opcões] comando [entradaDoComando]

DESCRIÇÃO

O comando xargs é usado para construir e executar linhas de comando a partir da entrada padrão. Os dados da entrada padrão servirão de argumento para o comando utilizado com o xargs.
Por exemplo:

```
$xargs rm
```

Esta linha irá executar o comando rm para cada dado digitado na entrada padrão.
Utilizando pipe para redirecionarmos a saída, poderíamos construir o seguinte comando:

```
$ls | xargs rm
```

Deste modo, cada arquivo ou pasta encontrada pelo ls seria utilizada como argumento para o comando rm.

NOTAS

Apesar de poderoso, é preciso entender o comando para saber quando usá-lo.
Por exemplo:

```
$ls | cat
```

O comando cat imprime dados da entrada padrão, neste caso com o uso do pipe, o cat irá imprimir os dados da saída do comando ls, que é o nome de todos os arquivos e pastas do diretório corrente. Então basicamente este comando é o mesmo que utilizar

```
$ls
```

Mas se utilizarmos o xargs:

```
$ls | xargs cat
```

A interpretação será outra. Neste caso o comando cat irá imprimir o conteúdo de cada arquivo encontrado pelo ls.

Comando 4.5 `tee` Envia conteúdo para saída padrão e arquivo simultaneamente.

NOME

tee

SINTAXE

`tee` [opcões] [arquivo]

DESCRIÇÃO

O comando canaliza a entrada padrão para o arquivo fornecido como argumento e também para a saída padrão. É normalmente utilizado com pipes para redirecionamento.
Por exemplo:

```
$ls | tee arquivos.txt
```

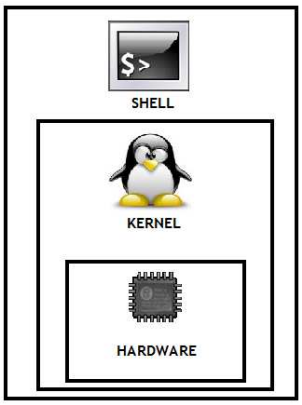
Neste caso os nomes dos arquivos e pastas encontrados pelo ls são impressos na tela e salvos no arquivo “arquivos.txt”.

4.2 Minha Pasta Pessoal

4.2.1 Shell

O Shell é um interpretador de comandos que possibilita a interação entre o usuário e o sistema, pois repassa ao kernel os comandos escritos pelo usuário ou comandos presentes em um arquivo executável.

As três camadas dum sistema baseado no kernel Linux são:



1. Shell, o interpretador que interage com o Kernel.
2. Kernel, o núcleo do sistema operacional que interage com o hardware (gerenciador de processos, gerenciador de memória , etc).
3. Hardware(HD, memória RAM, cdrom, etc), a camada mais interna.

O Shell padrão do GNU/Linux é o Bash, mas existem muitos tipos diferentes de Shell:

sh "Bourne shell" , escrito por Stephen Bourne. Ainda presente em muitos sistemas.

csh "C shell", escrito por Bill Joy. Possui sintaxe muito parecida com a linguagem de programação C.

ksh "Korn shell" , escrito por David Korn. Evolução do sh com comandos avançados.

bash "Bourne Again shell", escrito por . É uma extensão do sh. É o Shell padrão do sistem operacional GNU/Linux.

Antes de repassar a linha de comando para o kernel, o Shell executa alguns passos: substitui os caracteres especiais, alias e variáveis por seus possíveis valores; Verifica se o programa/função existe e depois quais são as permissões dos arquivos envolvidos; Chama o kernel para executar a linha de comando em um "shell-filho" ganhando um PID(número de processo); Permanece inativo até que o comando seja executado, a não ser que a execução seja feita em background(Ver subseção 4.3.4).

Se o shell não conseguir executar a linha de comando por não ter encontrado a função ou programa, devolverá uma mensagem de erro.

4.2.2 Algumas definições importantes

Um *shell interativo* é aquele pelo qual o usuário pode inserir uma linha de comando por vez, como o terminal X por exemplo.

Um *shell não interativo* é aquele invocado para que um shell script seja executado, por exemplo. É fechado assim que a execução termina.

Um *shell com login* é aquele que pede 'usuario' e 'senha' para que possa ser executado, ou que aquele que possui a opção '-login'.

Terminal ou *console*, é onde é mostrado o shell. Quando está em modo texto pode ser chamado também de Terminal Virtual. No Linux, é possivel alternar entre os terminais virtuais usando Alt + Fn, onde n é o número do terminal escolhido. Existem alguns programas que emulam o console no modo gráfico, como por exemplo o Xterm e o Eterm.

Prompt é a linha mostrada antes do espaço para digitar o comando e, por padrão, é formada da seguinte forma: nome do usuário que está acessando o Shell, seguido de um arroba, depois o nome da maquina e por último o diretorio atual, seguido de um $ (para o usuario normal) ou # (para o usuario root).

4.2.3 Shell Script

Daqui até o fim dessa seção, consideraremos que o shel utilizado seja o **bash**.

Um Shell Script é um arquivo que contém uma sequência de comandos e estruturas lógicas (if, for, while) - comumente encontradas em linguagens de programação. Serve para formarmos conjuntos de comandos que serão executados em sequência. Podemos, por exemplo, fazer um Shell Script para que um diretório seja criado , e dentro dele 2 arquivos texto. Esse arquivo poderá conter os comandos **cd**, **mkdir**, **touch**, **chmod**, dentre outros, e substituirá várias linhas de comando que, uma a uma, deveriam ser digitadas pelo usuário. Esse é um exemplo muito simples, pois o poder do Shell Script aumenta muito quando inserimos estruturas lógicas. Para que o shell reconheça o arquivo como um Shell Script, é necessário que na primeira linha do arquivo haja um *shbang* '#!' seguido do diretório onde se encontra o shell padrão. Geralmente criamos arquivos de Shell Script com extensão '.sh'.

Acompanhe um exemplo simples de Shell Script com comandos usuais do bash:

```
#!/bin/bash
# Função: Informa diretório corrente, lista arquivos e
# imprime mensagem
pwd
ls -l
echo "fim do script."
```

As linhas iniciadas por '#' são comentários, ou seja, não serão executadas. Acompanhe um exemplo mais elaborado de Shell Script, usando comandos do bash e estruturas lógicas de programação.

```
#!/bin/bash
# Autor: Vinicius Freire Função: soma 2 + 2 até chegar
# em 30
echo 'Deseja realizar uma operação de loop finita?[s / n]'
read resposta
if [ \$resposta = s ] then
  soma=0
  while ["\$soma" -le 30]
    do
  echo "\$soma"
  soma=‘expr \$soma + 2‘
done
echo "Que pena! Fim do script."
fi
```

O arquivo do Shell Script pode ser executado diretamente com o comando 'source', ou pode ser colocado em um arquivo de configuração, que é executado após o login.

4.2.4 Arquivos importantes de configuração

Existem muitos arquivos importantes de configuração usados quando o shell é inicializado.

Quando um shell interativo com login ou um shell não interativo com a opção -login é invocado, para o usuário root por exemplo, o primeiro arquivo executado é `/etc/profile`, que guarda informações de configurações para todos os usuários do sistema. Depois de executar os comandos desse arquivo, o shell busca um dos seguintes arquivos, nesta ordem: `$HOME/.bash_profile`, `$HOME/.bash_login` e `$HOME/.profile`, que guardam informações características do usuario que está logando. O primeiro que for encontrado e puder ser lido, será executado. Quando um shell com login é fechado, o shell procura pelo arquivo `$HOME/.bash_logout` e o executa, se ele existir.

Quando o shell é invocado como sendo interativo sem login, um terminal X por exemplo, o shell procura pelo arquivo `home/.bashrc`, que guarda características padrão para qualquer shell sem login.

Quando o shell é invocado como sendo não interativo, para executar um script por exemplo, procura pela variável $SHELL_ENV, que guarda o caminho do arquivo que deve ser executado para esse tipo de shell, se ela existir.

Os arquivos `$HOME/.bash_profile`, `$HOME/.bash_login`, `$HOME/.profile` guardam características do usuário atualmente logado - comandos que devem ser executados ao iniciar um novo shell, alias e variáveis definidas - por isso estão no diretório home de cada usuário.

Por que isso é importante? Porque podemos configurar coisas interessantes para cada usuário do sistema, que serão executados assim que o shell for invocado, ou no caso dos shells com login, fechado. Por exemplo, posso fazer com que todos os arquivos temporários sejam apagados quando eu me deslogar do sistema colocando alguns comandos no arquivo `$HOME/.bash_logout`, e toda vez que me deslogar, isso será executado. Posso determinar que para qualquer shell sem login (terminais X abertos por exemplo), a cor das letras da linha de comando será verde, e que para o usuário root, será azul. Posso determinar que determinados usuários usam o shell **`bash`**, e outros usam o shell **`csh`**, que alguns usam o **`vi`** como editor de texto padrão, outros usam o **`gedit`**. Posso determinar o email

padrão de cada usuário. Existem muitas opções para configuração do ambiente de cada usuário. Fazemos tudo isso modificando e criando aliases, variáveis e outros comandos nos arquivos de configuração correspondentes, que serão executados assim que o shell for invocado.

Outro arquivo importante é o `$HOME/.bash_history`, que guarda os últimos comandos digitados pelo usuário. O número de comandos guardados é definido pela variável `$HISTSIZE`.

4.2.5 Variáveis de ambiente

Em um computador, variáveis sao espaços da memória que guardam alguma informação. Existem muitas variáveis já definidas pelo sistema, como por exemplo `$HOST` e `$OSTYPE`, que guardam o nome do computador e o tipo de sistema operacional usado, respectivamente. Para o shell existem algumas variáveis especiais, como as citadas acima, chamadas variáveis de ambiente, porque armazenam características do ambiente do usuário. Essas variáveis existem para que quando o usuário logar no sistema, sempre encontre determinadas características, como por exemplo um determinado tipo de shell. Algumas dessas variáveis de ambiente guardam características definidas para todos os usuários. Elas se encontram no arquivo `/etc/profile`. Outras guardam características definidas para cada usuário do sistema, e podem estar nos arquivos `.bash_profile` ou `.bashrc`, presentes na home de cada usuário. As variáveis de ambiente ja definidas sao escritas com letra maiúscula e ao serem referenciadas, sao escritas com um $ na frente.

Algumas variáveis de ambiente do Linux:

$ARCH descrição da arquitetura da máquina;

$TERM tipo de terminal usado;

$HOST nome do computador;

$LANG código da língua padrão;

$PRINTER nome da impressora padrão;

$SHELL caminho do shell padrão;

$PATH lista de diretórios percorridos para executar comandos;

$OSTYPE nome do sistema operacional usado;

$HOME caminho do diretório do usuário atual;

$USER usuário atual;

$HISTSIZE número de comandos guardados no histórico;

$PS1 configuração do prompt do usuário atual.

Podemos criar uma nova variável com qualquer valor que nos possa ser útil, como por exemplo o caminho de um diretório, um valor numérico ou um texto.

```
$CAMINHO=~/Arquivos/Nomes
$cd $CAMINHO
```

Lembrando que não há espaço antes ou depois do '=' e que é uma boa prática escrever o nome da variável com letras maiúsculas. Essas novas variáveis estarão disponíveis enquanto o usuário estiver logado e usando o mesmo terminal, a nao ser que sejam criadas no arquivo `$HOME/.bash_profile` , `$HOME/.bashrc` ou `etc/profile`.

4.2.6 Alias

Muitas vezes é necessário escrever uma linha de comando muito grande várias vezes. Para facilitar essa tarefa, podemos dar um "apelido" para essa linha, ou seja, podemos criar uma maneira mais simples de chamar uma linha de comando útil. O comando **`ls`** é um exemplo de alias que vem como padrão do sistema. É interessante para o usuário perceber a lista gerada pelo **`ls`** dividida por cores conforme o tipo de arquivo, então ao digitar **`ls`** no terminal, na verdade o comando executado é **`ls -color`**. Esse é um tipo de alias permanente. Para tornar um alias permanente basta criá-lo diretamente no arquivo `$HOME/.bash_profile` , `$HOME/.bashrc` ou `etc/profile`, ou ainda criar um arquivo chamado `$HOME/.bash_aliases` apenas para guardar os aliases. Caso contrário, o alias será perdido ao se fechar o terminal.

Exercício Resolvido 4.3 *Crie um pequeno Shell Script que crie um novo diretorio.*

```
#!/bin/bash
# Script cria_diretorio.sh - cria diretório para guardar
documentos.
cd ~
NOME="$USER-Documentos"
mkdir $NOME
cd $NOME
echo "Olá, nesse diretório você pode guardar todos os seus
textos."
```

Para criar um Script em Shell, apenas devemos prestar atenção em alguns detalhes, como o shbang no inicio do script seguido do diretório do Shell padrão e a maneira como podemos fazer comentários. Também é bom lembrar que, embora o Linux não reconheca o arquivo pela extensão que está em seu nome, é bom criarmos o arquivo do script com a extensão .sh, e que para executá-lo basta usar o comando `source`. Segue um exeplo de um Shell Script que utiliza apenas comandos simples do Bash.

Exercício Resolvido 4.4 *Qual a diferença entre as duas linhas de comando abaixo?*

```
$ ./Script_cria_diretorio.sh
$ .Script_cria_diretorio.sh
```

A primeira linha executa o script, a segunda apenas é o nome de um arquivo oculto.

Comando 4.6 `source` Executa um script

NOME

source

SINTAXE

`source` arquivo
`.` arquivo

DESCRIÇÃO

Executa o conteúdo de arquivo como comandos do BASH. Ponto "." é um sinonimo para source.

Comando 4.7 `history` Mostra histórico de comandos.

NOME

history

SINTAXE

`history` [opções]

DESCRIÇÃO

Mostra histórico de comandos.

OPÇÕES

`-c` Apaga o histórico.

`-d n` Deleta a linha de numero "n" do histórico.

NOTAS

Pressionar flecha para cima recupera o ultimo comando usado.

EXEMPLOS

Para recuperar alguns comandos do histórico:

`$!30` Se refere ao comando da linha do histórico de número 30;

`$!!` Se refere ao comando anterior;

`$!echo`
Se refere ao comando mais recente que começa com "echo".

Comando 4.8 echo Mostra uma mensagem

NOME

echo

SINTAXE

echo string

DESCRIÇÃO

Imprime string na tela.

NOTAS

Um ponto interessante do `echo` se refere ao exit status de um comando, um valor que todo comando recebe após ser executado, onde 0 significa Tudo ok e um número diferente de 0 Você tem alguns problemas . Podemos recuperar esse valor com a variável `$?`. Dessa forma, será mostrado o exit status do último comando usado.

EXEMPLOS

Recuperando o valor de uma variável:

```
$echo $VARIAVEL
```

Comando 4.9 env Variáveis ambiente

NOME

env

DESCRIÇÃO

Altera as variáveis de ambiente para que certo comando seja rodado nas variáveis alteradas, depois o processo pode ser desfeito.

SINTAXE

env [opções] Variável=Valor Comando

Comando 4.10 set Manipula variáveis e funções do shell.

NOME

set

SINTAXE

set [opções] [argumento]

DESCRIÇÃO

Se nenhum argumento for passado ao comando, 'set' irá imprimir os nomes de valores das variaveis definidas e funções definidas.

OPÇÕES

`-a` Mostra variaveis que foram criadas ou modificadas pelo export.

Comando 4.11 `export` Determina uma variável de ambiente.

NOME

export

SINTAXE

`export` `[opções] [nome[=valor]]`

DESCRIÇÃO

Se nenhum valor for passado ao comando, 'export' irá imprimir todas variaveis exportadas.

Comando 4.12 `alias` Cria um alias

NOME

alias

SINTAXE

`alias` `[opções]` nome=valor

DESCRIÇÃO

O comando valor poderá ser executado atravéz de nome Se nenhum valor for passado ao comando, 'alias' irá imprimir todos os alias definidos. O nome do alias não pode ser "alias" ou "unalias". O nome do alias e o seu conteúdo podem conter metacaracteres, com excessão do "=". Para tornar um alias permanente, você pode criá-lo em um dos arquivos de configuração existentes em sua home, ou criar um arquivo chamado `home/.bash_aliases` para guardar todos os seus aliases.

EXEMPLOS

Um alias para intalação:

```
$ alias instalar=''sudo apt-get install''
```

Comando 4.13 `unalias` Exclui um alias

NOME

unalias

SINTAXE

`unalias` `[opções]` nome

DESCRIÇÃO

Apaga o alias nome.

OPÇÕES

`-a` Remove todos os alias.

4.3 Gerência de Processos

Um *processo* nada mais é do que um programa(comando) em execução. É fácil perceber que no Linux é possível que vários processos sejam executados ao mesmo tempo. Os processos possuem alguns atributos, usados pelo SO em sua tarefa de gerenciá-los. São estes alguns dos principais: identificador único (***P**rocess **ID**entifier*=*PID*), proprietário (***U**ser **ID**entifier*=*UID*) e estado.

4.3.1 PID e PPID

O identificador único de cada processo é um número. Às vezes é preciso duplicar um processo. Ao se fazer uma cópia, o processo copiado é chamado de *processo pai* e o novo de *processo filho*. O filho possui um identificador dizendo qual é o seu pai: *PPID* (***P**arent **P**rocess **ID**entifier*). Caso o processo pai seja interrompido, o mesmo acontece com todos os seus filhos. (Isso explica porque todos os programas abertos em uma sessão do terminal são encerrados quando ele é fechado.)

4.3.2 Estados

Os estados em que um processo pode se encontrar são:

`Em execução`
: Está sendo executado em primeiro ou segundo plano;

`Zumbi` Deveria estar morto, porém continua existindo por algum motivo;

`Parado`
: Teve sua execução interrompida mas pode ser reativado.

4.3.3 Sinais

Para interferir na execução dos processos ou comunicação entre estes, são enviados sinais. Após receber um, o processo realiza a tarefa a ele determinada, de acordo com o sinal. Alguns exemplos:

`STOP` Interrompe a execução de um processo para posterior reativação;

`CONT` Reativa um processo interrompido por STOP;

`TERM` Termina o processo, porém alguns são programados para ignorar este sinal;

`KILL` “Mata” o processo. Não pode ser ignorado.

4.3.4 Background e Foreground

Quando executamos um programa no terminal, o prompt de comando fica travado até o fim do processo. Chamamos isso de execução em primeiro plano, ou *foreground*. Como nem todos os comandos tem execução instantânea e alguns programas levam um certo tempo para finalizar, podemos enviar sinais de interrupção,

tornando possível a execução de outras tarefas na sessão atual do terminal. Para isso, usamos alguns atalhos de teclado:

`Ctrl`+`Z`
: Suspende o processo em execução atualmente (muda estado para "Parado");

`Ctrl`+`C`
: Termina a execução do processo.

Quando se suspende a execução de um processo, é possível voltar a executá-lo usando o comando (comando 4.19, p. 52). Com isso voltamos à situação inicial. Uma outra alternativa, é deixar a aplicação rodando em segundo plano, ou *background*. Fazemos isso usando o comando (comando 4.18, p. 51). A situação é como a ilustrada abaixo:

```
$ browser
```

Aqui o browser é chamado e a janela dele é aberta.

```
^Z
```

Em seguida suspendemos sua execução (Ctrl+Z), e a seguinte mensagem aparece:

```
[1]+ Stopped browser
```

[1] é o número de *job* do processo (*Jobs* não serão abordados nesta apostila. Para mais informações veja **`man jobs`**) que é usado para identificar qual comando colocar em *background* ou *foreground*. *Stopped* é o novo *status* do processo (Parado). E *browser* é o nome da aplicação.

```
$ bg
```

Agora o colocamos para executar em *background*. Por padrão o comando **`bg`** se aplica ao último processo suspenso. E a seguinte mensagem é exibida:

```
$ [1]+ browser &
$
```

Novamente temos o número de *job* e o nome, mas agora com uma novidade, o &, que será explicado em seguida.

Finalmente, você pode continuar navegando no browser e executando comandos no terminal.

Outra maneira, mais direta, de se rodar o programa em segundo plano é acrescentar o símbolo & ao final do comando. A execução procede da seguinte maneira:

```
$ browser &
[3] 5940
$
```

Nesse exemplo, 5940 é o `PID` do processo.

Comando 4.14 `ps` Retorna os processos correntes.

SINTAXE

`ps [opções]`

DESCRIÇÃO

Por padrão, exibe os processos que possuem mesmo UID que o usuário corrente e associados com o mesmo terminal de onde o comando foi executado. A vizualisação mostra PID, terminal ao qual está associado (TTY), tempo de CPU acumulado(TIME) e o nome da aplicação(COMMAND), sem seguir nenhuma ordenação.

OPÇÕES

`a` mostra todos os processos existentes;

`e` exibe as variáveis de ambiente relacionadas aos processos;

`f` exibe a árvore de execução dos processos;

`l` exibe informações mais detalhadas dos processos;

`m` mostra a quantidade de memória ocupada por cada processo;

`u` exibe o nome do usuário que iniciou determinado processo e a hora em que isso ocorreu;

`x` exibe os processos que não estão associados a terminais;

`w` se os detalhes de um processo não couber em uma linha, essa opção faz com que o restante seja exibido na linha seguinte.

USOS FREQUENTES

`ps aux` Exibe todos os processos com nome de usuário e data.

VEJA TAMBÉM

(comando 4.15, p. 50).

Comando 4.15 `top` Exibe as tarefas do Linux.

SINTAXE

`top`

DESCRIÇÃO

Exibe, em tempo real, informações sobre seu sistema Linux, processos em andamento e recursos do sistema, incluídos CPU, memória RAM e uso do swap, além do número total de tarefas sendo executadas.
Por padrão, sobre cada processo são exibidos vários campos. Alguns deles:

`PID` O identificador;

`USER` O proprietário;

`PR` Número de prioridade;

`VIRT` Volume de *memória virtual* usada pelo processo;

`RES` Quantidade de *memória física não-swap* utilizada;

`%CPU` Porcentagem de uso de CPU

`%MEM` Porcentagem de uso de Memória Física.

`TIME+`
Tempo total de CPU usado desde o início do processo. Se estiver em "Modo cumulativo"(Ver comandos interarivos abaixo) Exime o tempo de CPU que o processo e seus "filhos"já terminados usaram.

`COMMAND`
Nome do programa ou linha de comando.

O top também nos permite a manipulação dos processos por meio de comandos interativos. Alguns dos mais importantes:

`k` Solicita um PID e finaliza o processo correspondente.

`M` Ordena os processos pelo uso da memória residente.

`N` Ordena os processos pelos seus PID's.

`P` Ordena os processos pelo uso da CPU (esta é a ordenação padrão deste comando).

`espaço`
Atualiza a visualização do quadro de processos.

`S` Alterna a exibição de tempo para modo acumulativo.

`s` Solicita um novo intervalo de tempo entre cada atualização.

`h` Exibe a ajuda dos comandos interativos.

`q` Aborta o comando.

VEJA TAMBÉM

(comando 4.14, p. 49), pstree.

Comando 4.16 `kill` Manda um sinal para um processo a partir de seu PID.

SINTAXE

```
kill [-sinal] PID ...
kill -l [-sinal] ...
```

DESCRIÇÃO

Por padrão o comando envia um sinal TERM (se nenhuma opção de sinal for escolhida). O kill aceita 3 formas de se especificar os sinais; o kill por exemplo pode ser: -KILL, -SIGKILL e -9. Para saber a numeração de cada sinal, veja **`man kill`**.

`-l` - Por padrão, exibe uma lista de sinais disponíveis. Se especificar um número, converte-o para o nome do sinal e vice-versa.

EXEMPLOS

`kill -9 -1` Mata todos os processos que você pode matar.

VEJA TAMBÉM

(comando 4.17, p. 51)

Comando 4.17 `pkill` Envia um sinal para processos a partir de nome e outros atributos.

SINTAXE

```
pkill [-sinal] [-opção atributo, ...]  [expressão]
```

DESCRIÇÃO

Por padrão envia o sinal TERM para cada processo e o campo expressão procura por nome do processo. Para selecionar processo por outro atributo além do nome, usa-se uma opção e um filtro desejado (por exemplo: processos do usuário pet).

`-u UID, ...`
Filtra por processos cujo UID real corresponde ao fornecido. Aceita tanto UID como nome.

`-t terminal, ...`
Filtra por processos controlados pelo terminal fornecido.

VEJA TAMBÉM

(comando 4.16, p. 51), pgrep, (comando 4.14, p. 49).

Comando 4.18 `bg` Roda tarefas em *background*.

SINTAXE

```
bg [job_id], ...]
```

DESCRIÇÃO

Por padrão move o processo com maior número de job para *background*.

Comando 4.19 `fg` Roda tarefas em *foreground*.

SINTAXE

```
fg [job_id], ...]
```

DESCRIÇÃO

Por padrão move o processo com maior número de job para *foreground*.

Comando 4.20 `nohup` Executa o comando de forma imune ao sinal HUP.

SINTAXE

```
nohup comando [argumentos], ...
```

DESCRIÇÃO

Evitando sinais HUP, o processo continua rodando mesmo após o usuário ter se desconectado do sistema.

VEJA TAMBÉM

`man nohup`.

4.4 Sistemas de Arquivos

São as estruturas lógicas usadas para organizar o armazenamento os dados no disco rígido, de maneira a permitir que o sistema operacional possa controlar o acesso ao disco. A lista de sistemas de arquivos suportada pelo Linux é extensa, sendo atualmente os principais: *ext3* e *ReiserFS*. *ReiserFS* é em geral mais rápido e melhor para partições com maior número de arquivos menores e *ext3* com arquivos maiores. Então, para poder acessar os dados dos diversos dispositivos (Disco rígido, CD-ROM's, pen-drives, ...) é preciso montar um sistema de arquivos. Ao fazer isso, os dados se tornam disponíveis dentro de um diretório que representará o dispositivo. Algumas distribuições Linux detectam e montam novos dispositivos (geralmente externos) automaticamente, porém, usuários sem permissões de super usuário não podem montar sistemas de arquivos manualmente.

Exercício Resolvido 4.5 *Como matar um processo de* ***PID*** *3389?*

```
$kill 3389
```

É muito simples, basta usar o comando ***kill****. Como argumento desse comando, podemos usar o PID de cada processo, que pode ser obtido com o comando* ***ps****.*

Comando 4.21 du Estima o uso de espaço de arquivos.

SINTAXE

du [opção] [arquivo] ...

DESCRIÇÃO

Resume o uso de disco de cada arquivo, *recursivamente* no caso de diretórios. Por padrão usa o arquivo como sendo o diretório atual, exibindo o tamanho geral e o caminho de cada diretório filho na árvore de hierarquia.

OPÇÕES

`-a` Exibe o tamanho de todos os arquivos, não somente diretórios.

`-b` Exibe o tamanho em bytes.

`-h, --human-readable`
Exibe o tamnho usando quantificadores(1K, 234M, 2G, ...)(forma "legível para humanos").

Comando 4.22 `df` Mostra o uso de espaço em disco de sistema de arquivos.

SINTAXE

`df [opção] [arquivo] ...`

DESCRIÇÃO

df mostra o espaço em disco disponível no sistema de arquivo. Se não for especificado nenhum, mostra o espaço disponível em todos os sistemas atualmente montados. O espaço em disco é mostrado em blocos de 1K, por padrão.

OPÇÕES

`-h, --human-readable`
Exibe o tamnho usando quantificadores(1K, 234M, 2G, ...)(forma "legível à humanos").

`-l, --local`
Limita exibição para sistemas de arquivos locais.

`-T, --print-type`
Exibe o tipo de sistema.

Comando 4.23 `stat` Mostra informações sobre arquivos.

NOME

stat

SINTAXE

`stat` `arquivo ...`

DESCRIÇÃO

Este comando mostra o formato do arquivo, o tamanho, a posição no HD, as permissões, o login e o grupo do dono e os tempos *access*, *modify* e *change*.

OPÇÕES

`-f, --file-system`
Mostra *status* de um sistema de arquivos ao invés de um arquivo.]

Comando 4.24 `mount` Monta um sistema de arquivos.

SINTAXE

`mount [opções] dispositivo diretório`

DESCRIÇÃO

Monta um dispositivo no diretório especificado. Por padrão, sem nenhuma opção, mostra todos os dispositivos montados atualmente.

OPCOES

`-a` Monta todos os dispositivos encontrados no arquivo `/etc/fstab` .

`-r` Monta o dispositivo em modo somente leitura.

`-t` Usado para especificar qual o tipo de sistema de arquivos a ser montado. Os tipos suportados atualmente são:adfs, affs, autofs, cifs, coda, coherent, cramfs, debugfs, devpts, efs, ext, ext2, ext3, ext4, hfs, hfsplus, hpfs, iso9660, jfs, minix, msdos, ncpfs, nfs, nfs4, ntfs, proc, qnx4, ramfs, reiserfs, romfs, smbfs, sysv, tmpfs, udf, ufs, umsdos, usbfs, vfat (fat32), xenix, xfs, xiafs.

EXEMPLO DE USO

`mount /dev/fd0 /mnt/floppy` Torna acessível o sistema de arquivos encontrado no dispositivo `/dev/fd0` (normalmente o disquete) no diretório `/mnt/floppy`.

VEJA TAMBÉM

(comando 4.25, p. 55).

Comando 4.25 `umount` Desmonta um sistema de arquivos.

SINTAXE

`umount [opções] dispositivo | diretório`

DESCRIÇÃO

Desmonta um sistema de arquivos, especificando o diretório onde foi montado ou o dispositivo. O umount não executa se o sistema de arquivos estiver ocupado (executando alguma operação).

`-a` Desmonta todos os sistemas de arquivos descritos em `/etc/mtab` .

`-t` Indica que as ações devem apenas ser tomandas em sistemas do tipo especificado. Mais de um tipo pode ser usado, fazendo-se uma lista separada por vírgulas.

`-f` Força o sistema a ser desmontado.

VEJA TAMBÉM

(comando 4.24, p. 55).

4.5 Exercícios

Para resolver esses desafios, utilize as man pages, juntamente com as informaçoes da apostila. Não esqueça de fazer o download do diretório de exercicios que acompanha a apostila.

Exercício 4.6 *Crie um novo arquivo com as mesmas informações do arquivo*

```
pasta\_exercicios/Poesias/PoemaLeminski.txt
```

porém que contenha o número de cada linha no início da linha, apenas pelo terminal.

Exercício 4.7 *Observando a seguinte linha de comando no diretorio*

```
$ ls pasta\_exercicios/Diversos/novo_diretorio.tx
novo_diretorio.txt
$ cat novo\_diretorio.txt | xargs mkdir
```

De que maneira é possivel modificar essa linha de modo que seja pedido ao usuário a confirmação para se criar o diretório `/pasta_exercicios/Diversos/Textos`*?*

Exercício 4.8 *Usando apenas o comando* **`less`**, *execute os seguintes passos: abra o arquivo* `pasta_exercicios/Empresa/produtos.txt` *com o comando* **`less`** *e busque as palavras da lista abaixo. A cada palavra, encontre um jeito de marcar essa posição, referenciando-a com a letra maiúscula correspondente. Se necessário, volte ao início do arquivo para encontrar todas as palavras. Depois disso, vá para o final do arquivo e encontre uma maneira de voltar em todas as marcações que você fez, sem fazer referência à palavra que havia sido buscada para essa marcação, mas sim referenciando a letra correspondente. Verifique se você está mesmo no lugar certo do arquivo (a posição em que aparece a palavra correspondente), fazendo uma nova busca pela palavra referenciada por aquela letra.*

Palavras: HDMI(A), PENTIUM(B), ANTENA(C), ECOPOWER(D)

Exercício 4.9 *Cria um alias chamado site_interessante, que toda vez que for executado, abrirá um site específico pelo Firefox.*

Exercício 4.10 *Configure seu prompt de modo que ele contenha a sequência de caracteres "=D" na cor roxa, seguido do nome do usuário na cor branca e por último, o caracter "¿". Faça com que essa configuração seja permanente.*

Exercício 4.11 *Recuperando a variável $?, só conseguimos recuperar o valor de exit status do último comando executado, portanto embora uma das linhas de comando abaixo esteja errada, o valor da variavel $? é 0, pois o comando* **`tee`** *é executado corretamente nas duas linhas . Como é possível recuperar o valor de exit status dos outros comandos dessas linhas?*

```
\$ whoami | xargs finger | tee info.txt
\$ echo \$?
0
\$ whoami | xrg finger | tee info.txt
\$ echo \$?
0
```

Exercício 4.12 *Imprima na tela todos os processos do sistema, juntamente com a coluna de valor STAT, que contém a situação do processo. Observe a linha que*

contém informações sobre o comando que você acabou de executar. O que significa o valor da coluna STAT dessa linha?

Exercício 4.13 *Qual a diferenca entre o comando* **`ps`** *e o comando* **`top`***?*

Capítulo 5

Terminal do Linux, Uma Mão na Roda

5.1 Busca de Arquivos

Conforme usamos um sistema operacional é muito comum acumularmos diversos arquivos pessoais. Mesmo mantendo uma boa organização, se ficarmos muito tempo sem acessar um determinado arquivo ou tivermos milhares de diretórios e arquivos em nossa pasta pessoal, é bastante provável que tenhamos problemas ao procura-los.

Para auxiliar o usuário neste tipo de busca, o Linux nos oferece uma ótima ferramenta: o comando **`find`**, que permite que façamos buscas por inúmeras características como nome, tipo de arquivo, permissões, datas de criação, modificação ou último acesso ao arquivo.

Com o uso correto do **`find`** a tarefa de encontrar um arquivo qualquer no sistema fica fácil, porém certos cuidados devem ser tomados. Como o find realiza buscas recursivas, percorrer uma árvore de diretórios muito extensa pode ser demorado em demasia. Além disso, ao tentar buscar por diretórios em que não posuímos permissões ele emite *muitas* mensagens de erro. Nestes casos é recomendado enviar todas as mensagens de erro para um arquivo ou para `/dev/null` (conforme seção 4.1.4). Vale também ressaltar que na busca por nomes o **`find`** utiliza expressões regulares.

Um outro comando importante, é o **`whereis`**, que nos ajuda a localizar em qual diretório se encontra um executável e onde fica armazenada sua página de manual. Com o bom uso destes dois comandos, localizar dados torna-se rápido, e eficiente.

Comando 5.1 `find` Busca por arquivos, parte 1.

NOME

find

SINTAXE

`find` `[Endereco] Argurmento`

DESCRIÇÃO

O comando find procura por itens no sistema. O endereço é por onde ele irá começar a busca e o argumento define qual informação do item, ele irá usar para realizar a busca.

ARGUMENTO

`-name Expr.`
Procura os itens pelo nome do arquivo;

`-iname Expr.`
Procura pelo nome do arquivo, ignorando a caixa;

`-perm Permissao`
Procura pela permissao do arquivo;

`-type Tipo`
Procura por arquivos comum;

`-atime Tempo`
Procura pelo tempo *access* do arquivo;

`-ctime Tempo`
Procura pelo tempo *change* do arquivo;

`-mtime Tempo`
Procura pelo tempo *modify* do arquivo;

LEGENDA

`Expr.` Qualquer expressão aceita pelo console, veja mais na sessão .....;

`Permissao`
-[gou]=[wrx] ou em octal (-777);
procurar por arquivos, ...
onde o usuario tem permissao de leitura: -u=r;
onde o usuario e o grupo tem permissao de escrita -ug=w;
onde o outros tem permissao de escrita e leitura -o=rw;

`Tipo` d (diretorio) ou f (arquivo comum) ou l (link simbolico);

`Tempo` [+-]n * 24horas;
procurar por arquivos, ...
antes de 3 dias atras: -3;
depois de 6 dias atras: +6;
exatamente a 2 dias atras: 2;

EXEMPLOS

Procura um item com nome arquivo.txt e começa a busca no diretorio /home/usuario
find /home/usuario -iname ”arquivo.txt”

Procura um diretorio e começa a busca no diretorio HOME
find -type d

Procura um item com nome ”xorg.*”, lembrando que * pode ser qualquer coisa, e começa a busca no diretorio /etc
find /etc -iname ”xorg.*”

Comando 5.2 `find` Busca por arquivos, parte 2.

NOME

find

SINTAXE

`find` `[Endereco] [OPCOES] Argurmento1 [[OP] [Argumento2] [OP] [Argumento3]...]`

DESCRIÇÃO

O comando find também pode ser utilizados com vários argumentos para refinar a busca.

OPCOES

`-mindepth N`
: Define a altura minima da árvore para começar a procurar;

`-maxdepth N`
: Define a altura maxima da árvore para terminar a busca;

Obs: Normalmente utilizados essas opções para agilizar a busca.

OP

`-a` Argumento1 e Argumento2;

`-o` Argumento1 ou Argumento2;

OBS: Caso oculte o campo OP, será utilizado por padrão a opção -a;

EXEMPLOS

Procura por um item do tipo diretorio e com o nome ”fotos”
find -type d -a -iname ”fotos”

Procura por arquivos, que o grupo tem permissão de leitura ou de escrita
find -perm -g=r -o -perm -g=w

Procura por arquivos, que o grupo tem permissão de leitura ou de escrita e limita a busca no maximo de 2 subdiretorios
find -maxdepth 2 -perm -g=r -o -perm -g=w

OBSERVAÇÃO

As vezes é interessante redirecionar a saída de arquivos para /dev/null, pois existem muitas pastas no sistema, que não permite a sua visualização e, portanto, o resultado estará no meio de vários erros de permissões, o que atrapalha a visualização do resultado.

Comando 5.3 `find` Busca por arquivos, parte 3

NOME

find

SINTAXE

`find` `Endereco [OPCOES] Argumentos...  -exec` Comando {}

DESCRIÇÃO

O comando find junto com a opção -exec procura itens no sistema e para cada item encontrado executa o Comando, que receberá o nome do item através do .

EXEMPLOS

Procura por arquivos, que tem a extensão txt e para cada item encontrado executa o comando file ItemEncontrado.
find -iname *.txt -exec file ;

Comando 5.4 `locate` Busca arquivos pelo nome.

NOME

locate

SINTAXE

`locate` arquivo

DESCRIÇÃO

O comando locate procura por arquivos assim como o comando find, mas difere por ter uma sintaxe mais fácil, porém mais restrita. Ele busca pelo nome do arquivo através de um banco de dados que contém as localizações dos arquivos.

Comando 5.5 `whereis` Busca por arquivos pertencentes ao sistema.

NOME

whereis

SINTAXE

`whereis` [opções] arquivo

DESCRIÇÃO

O comando whereis procura por manuais, programas ou codigo-fontes ligados ao sistema operacional. Por exemplo, para obter informações sobre o comando **`ls`** pode-se usar a seguinte linha:

```
$ whereis ls
```

OPÇÕES

`-b` Procura somente por programas;

`-m` Procura somente por manuais;

`-s` Procura somente por codigo-fontes;

Comando 5.6 `which` Retorna a localização dos comandos do sistema.

NOME

which

SINTAXE

`which` comando

DESCRIÇÃO

O comando which retorna a localização de algum comando do sistema operacional. Similar ao **`whereis -b`** porém, mais objetivo.

5.2 Busca Dentro de Arquivos

Uma tarefa muito frequente que realizamos, principalmente quando estamos diante de listas, é localizar certas palavras. Esta tarefa é simples se pensarmos que nossa lista contém dezenas de itens, como o nome e telefone do pessoal do trabalho, por exemplo, e mais simples ainda se esta lista estiver ordenada. Os problemas começam a aparecer conforme aumenta o tamanho do texto em que buscamos as palavras, e principalmente quando não se tem uma regularidade apresentada no documento, como colunas que separam informações específicas ou uma ordenação sobre os dados, tornando assim a tarefa manual inviável. Existem ferramentas que realizam buscas sobre textos e indicam a localização de suas ocorrências, isto não é novidade, observamos isso, por exemplo, em diversos editores de texto e navegadores *web*.

Mas se ao invés de procurarmos por uma palavra, procurarmos por um conjunto de palavras com determinadas características? Podemos procurar apenas números de telefones, ou então, *e-mails* que aparecerem em um arquivo de texto, por exemplo. Vamos chamar estas características, seja elas quais forem, de padrão de busca, e para representá-lo utilizaremos as *expressões regulares*.

5.2.1 Expressões Regulares

Expressão regular é uma representação para um conjunto de *strings*, e é escrita utilizando-se alguns caracteres que possuem significados especiais, também chamados de operadores, além dos caracteres comuns, que mantem seu significado literal. Assim, `arara-azul` é uma expressão regular que casa com qualquer trecho que **contenha** exatamente essa sequência de caracteres: um `a`, depois um `r`, depois outro `a` ...

Vamos aos caracteres especiais, listados abaixo, e às suas funcionalidades, mas antes só um detalhe: para utilizar esses caracteres com seu significado literal, coloca-se uma contrabarra precedente. Para encontrarmos uma quantia em Reais num documento de uma forma bem simples, poderíamos fazer isso, `R\$`, por exemplo.

```
.    [ ]    ^    $    ?    *    +    { }    |    ( )
```

Caractere é a unidade de uma expressão regular e podemos representá-la da seguinte maneira:

`c` O caracter `c` casa com ele mesmo;

`.` O ponto casa com qualquer caracter;

`[ ]` Os colchetes formam uma lista, que casa com qualquer caractere que estiver contido nela. Uma lista que inicia com um acento circunflexo `[^ ]` é uma lista negada, ou seja, casa com qualquer caracter que não estiver contido nela. Dentro de uma lista, caracteres especiais perdem seu significado, com exceção do acento circunflexo na primeira posição, que indica uma lista negada, e do hífen `-`, que entre dois caracteres indica um intervalo. Assim `[a-dACE]` casa com as letras minúsculas `a`, `b`, `c` e `d` e também com as maiúsculas `A`, `C` e `E`.

Podemos ter em expressões regulares conjuntos, formados por uma ou mais unidades e também operadores. Estes operadores, listados abaixo, tratam do números de repetições dos elementos que os precedem. Chamaremos itens estes elementos, que podem ser unidades ou conjuntos, e veremos na sequência como representá-los.

? A interrogação torna o item que a precede opcional;

***** O asterisco faz com que o item que o precede ocorra zero ou mais vezes;

+ O mais faz com que o item que o precede ocorra uma ou mais vezes;

{n} As chaves fazem com que o item que as precedem ocorra n vezes.

Para representarmos uma lista de conjuntos, utilizamos a **|** (barra vertical) como separador, dentro das expressões contidas entre parênteses. Assim como uma lista entre colchetes, que casa somente um entre os caracteres listados, esta nova lista também só casa com um dos seus conjuntos.

(exp**)** Os parênteses transformam a expressão exp em um conjunto.

(exp**|**exp**|...)**
A barra vertical entre as expressões forma uma lista de conjuntos, que casa com qualquer uma das expressões exp.

Para encerrar a apresentação dos caracteres especiais, apresentaremos duas âncoras, que só podem ser colocadas no início ou no fim de uma expressão regular.

^ O acento circunflexo é colocado no início da expressão e casa com o início da linha;

$ O cifrão é colocado no fim da expressão e casa com o fim da linha.

5.2.2 Expressões no Terminal

Até aqui vimos a notação das expressões regulares. Esta notação vista é chamada no Linux de *expressão regular estendida*, que diferencia-se da *expressão regular básica* na maneira de representação de alguns caracteres especiais.

Para escrevermos expressões regulares básicas, os seguintes caracteres especiais devem ser precedidos por contrabarras:

```
?     +     {     |     ( )

\?    \+    \{    \|    \( \)
```

Comando 5.7 `grep` Busca por padrões de caracteres em cada linha do texto de entrada.

NOME

grep, egrep, fgrep

SINTAXE

`grep` [opções] padrão [arquivo] ...

DESCRIÇÃO

`Grep` busca pelo padrão de caracteres no arquivo especificado ou na entrada padrão se este parâmetro estiver ausente, e imprime as linhas que casam com este padrão.
As alternativas **`grep`**, **`egrep`** e **`fgrep`** se diferenciam no modo de interpretação do padrão a ser casado.

`grep` trata este padrão como uma expressão regular básica;

`egrep` trata este padrão como uma expressão regular estendida e é equivalente a **`grep -E`**;

`fgrep` trata este padrão como a sequência de caracteres é literalmente, sem interpretar nenhum caracer como especial, e é equivalente a **`grep -F`**.

OPÇÕES

`-i` Ignora a diferença entre caracteres maiúsculos e minúsculos na busca.

`-v` Inverte a lógica da busca, imprimindo todas as linhas que não contenham o padrão de busca.

`-c` Conta o número de ocorrências do padrão buscado, ao invés de imprimir as linhas que contenham o padrão.

`-o` Imprime somente a string que casou com o padrão, uma por linha, ao invés de imprimir a linha inteira.

`-n` Coloca a numeração nas linhas que casaram com o padrão.

`-A` NUM
Imprime as NUM linhas anteriores à linha casada inclusive ela mesma.

`-B` NUM
Imprime as NUM linhas posteriores à linha casada inclusive ela mesma.

`-C` NUM
Imprime as NUM linhas anteriores e posteriores à linha casada inclusive ela mesma.

5.3 Caixa de Ferramentas

Um dos grandes atrativos do sistema operacional Linux, é a possibilidade de realizar diversas operações sobre volumes significativos de dados de forma organizada e eficiente. O sistema oferece por padrão diversos aplicativos interessantes para automatizar operações sobre arquivos, que permitem a realização de muitas tarefas simples como contar o número de bytes, caracteres, palavras e linhas de um arquivo ou entrada padrão, além de nos ajudar (e muito) em tarefas um pouco mais complexas, através da combinação de comandos.

Com os comandos apresentados nesta seção, é possível filtrar conteúdo de arquivos, eliminando colunas, ordenando resultados, mudando a formatação, substituindo caracteres, entre outras coisas. Essas ferramentas são extremamente poderosas para lidar com grandes massas de dados, facilitando muito tal tarefa.

5.3.1 Manipulação de Linhas

Uma tarefa muito comum quando estamos lidando com arquivos através do terminal do Linux é a contagem de linhas e para isso temos o comando **`nl`** (comando 5.11, p. 76). Além deste ser um comando útil ao lidarmos com arquivos, ele pode ser combinado com a saída de outros comando. Podemos, por exemplo, combinar o **`nl`** com o comando **`ls -l`** (comando 3.10, p. 19) para contar o número de arquivos existentes em um diretório.

Apesar do comando **`nl`** nos ajudar na realização de diversas tarefas ele é limitado, pois conta apenas o número de linhas e nada mais. Existe um comando similar porém mais versátil, o comando **`wc`**. Através das opções do **`wc`**, podemos contar além do número de linhas o número de palavras, bytes ou caracteres de um arquivo ou saída de comando, e com isso ganhamos diversas novas funcionalidades.

Vamos agora veremos as ferramentas para selecionar linhas de um texto. Veremos dois comandos que permitem visualizar as `n` primeiras ou as `n` últimas linhas de um arquivo. Por exemplo, para visualizar as 10 primeiras linhas do arquivo `texto.txt` poderíamos utilizar a seguinte linha de comando:

```
\$ cat texto.txt | head -n 10
```

O comando **`head`** mostra as `n` primeiras linhas do arquivo ou da entrada padrão, e se especificarmos um número negativo de linhas, ele vai imprimir todas menos as `n` últimas linhas, agindo como o comando **`tail`** ao contrário. O comando **`tail`**, mostra as `n` últimas linhas ou todas menos as `n` primeiras linhas se colocarmos um + na frente do número:

```
\$ cat texto.txt | tail -n +5
```

Por exemplo, um arquivo que possua 30 linhas. Para imprimir as suas 10 primeiras pode-se usar:

```
\$ cat texto.txt | head -n 10
```

Ou pode-se pensar em imprimir todas as linhas menos as 20 últimas, o que neste caso, obteria o mesmo resultado.

```
\$ cat texto.txt | head -n -20
```

Um pensamento análogo funciona para o comando **tail**.

5.3.2 Seleção de Colunas

Quando estamos lidando com dados armazenados em arquivos é muito comum organiza-los em colunas, usando espaços ou outros caracteres como : (dois pontos) para separá-las. Esse modelo também aparece na saída de alguns comandos. Exemplo dessa formatação é o arquivo de sistema que é responsável pelos grupos: `/etc/group`.

O comando **cut** é muito útil para lidar com esse modelo em colunas, pois ele permite que você filtre um arquivo ou saída de comando de forma que apenas algumas colunas sejam exibidas. Seu uso é bastante simples, basta especificar qual o caractére responsável pela separação das colunas e quais colunas você quer que sejam mostradas, contando a partir do número 1. Por exemplo, para filtrar o conteúdo do arquivo `/etc/group` para que apenas a segunda coluna seja mostrada na tela devemos utilizar o seguinte comando:

```
\$ cut /etc/group -d":" -f2
```

O comando **cut** é muito simples, não tem mistério. Através da opção `-d` indicamos qual é o caractere que separa as colunas e através da opção `-f` indicamos quais colunas queremos visualizar. A maior utilidade do comando **cut** é usá-lo em conjunto com a saída de algum outro comando (ver 4.1.3).

5.3.3 Ordenação

Ao lidarmos com listas (cadastros de nomes, telefones, entre outros) é, em geral, muito interessante manter essa lista ordenada. Para ordenar uma lista o comando usado é o **sort**. Este comando permite a ordenação de uma lista de linhas de texto em ordem alfabética ou ainda, ordenar uma lista numericamente, além de oferecer algumas opções úteis como ordenação invertida e até “ordenação aleatória”.

5.3.4 Operações Aritméticas

Existe uma aplicação do sistema Linux chamada **bc** que é uma calculadora de precisão utilizada através da linha de comando. Ela é extremamente útil para a escrita de alguns scripts, mas também pode ser utilizada para operações em geral.

Existem duas principais formas de utilização do **bc**. A primeira é mais simples, basta digitar **bc** na linha de comando para ter acesso ao prompt do **bc** e então digitar as operações a serem realizadas.

A segunda forma é mais utilizada em scripts, mas também pode ser usada na linha de comando. Ela consiste em utilizar o comando **echo** para gerar a linha com as operações desejadas e passar isso para o **bc** através do `pipe`. Por exemplo, para executar a operação ”10+2”:

```
\$ echo 10+2 | bc
```

Além do **bc** efetuar operações padrões (divisão, multiplicação, soma e subtração), é possível realizar diversas operações mais complexas envolvendo variáveis e sentenças lógicas.

5.3.5 Substituição de caracteres

O que aparece com certa frequência quando estamos trabalhando com o terminal do Linux, é a necessidade de realizarmos substituições de caracteres, seja num arquivo, ou no resultado de um comando. Através do comando **tr** podemos fazer isso. Por exemplo, se quisermos substituir toda ocorrência do caractere a de uma frase por um e teríamos que executar:

```
\$ cat texto.txt | tr a e
```

O **tr** só lê da entrada padrão, e só escreve na saída padrão. Ele não abre arquivos diretamente, é sempre preciso usar o comando **cat** (comando 4.1, p. 36) para passar um arquivo para o **tr**.

Além de executar substituições de caracteres, o **tr** tem algumas outras funcionalidades interessantes, como a possibilidade de deletar caracteres e eliminar repetições.

5.3.6 Manipulação avançada de texto

SED

O **sed** é o comando de edição de fluxo (**S**tream **ED**itor) do Linux, e permite a realização de diversas tarefas de transformação de textos de forma automática. Além disso, sua versatilidade é grande, tanto que existem até jogos escritos através dele.

Usando o sed

O **sed** funciona como um filtro, especifica-se algumas regras de transformação do texto, ele recebe o texto de entrada, aplica as regras e joga o texto resultante na saída padrão. As regras do **sed** sempre seguem a seguinte sintaxe:

sed [opções] regras [arquivo]

E as regras são da seguinte forma:

[endereço 1[, endereço 2]] função [argumento]

O **sed** analisa linha a linha, de cima para baixo, da esquerda para a direita. O endereço serve para aplicar a função apenas às linhas indicadas, que podem ser especificadas pelo número da linha ou por parte do seu conteúdo entre / (barras) (O **sed** utiliza expressões regulares nessa busca por padrões, o que o torna mais poderoso ainda). Vamos ver alguns exemplos de endereçamento com a função p, que imprime linhas:

```
# Sem endereços, imprimindo todas as linhas:
\$ cat texto.txt | sed p

# Endereçamento direto, imprimindo a linha 5:
\$ cat texto.txt | sed '5p'

# Endereçamento direto, imprimindo da linha 5 até a linha 10:
\$ cat texto.txt | sed '5,10p'

# Endereçamento por conteúdo, imprimindo as linhas que contêm a palavra
# "abobrinha"
\$ cat texto.txt | sed '/abobrinha/p'

# Endereçamento direto, imprimindo da linha 5 até a primeira linha contendo
# a palavra "abobrinha":
\$ cat texto.txt | sed '5,/abobrinha/p'
```

Se você executou os comandos acima percebeu que as linhas são duplicadas, isso acontece porque por padrão o **sed** joga para a saída padrão todas as linhas do arquivo, o que geralmente causa alguma confusão (e nesse caso gera duplicação de linhas). Para resolver este problema basta chamar o **sed** com a opção `-n`, que faz com que o **sed** não imprima nada automaticamente na saída padrão. Se adicionássemos este parâmetro nos exemplos acima, apenas as linhas especificadas para a função `p` seriam impressas.

Outra característica importante do **sed** é a possibilidade de executar diversas funções ao mesmo tempo, separando-os por ponto-e-vírgula:

```
# Imprimir todas as linhas contendo "abobrinha",
# assim como as linhas de 1 a 10:
\$ cat texto.txt | sed '1,10p;/abobrinha/p'
```

Agora que já conhecemos a estrutura básica do comando **sed** vamos ver algumas outras funções importantes, começando com a função `s`, que realiza a substituição de padrões encontrados. A função `s` do sed é composto por duas partes que podem ser divididas por qualquer caractere, para que seja possível usar um separador que não aparece no padrão buscado ou no texto de substituição, por exemplo, se quiséssemos substituir um diretório, o caractere `/` não seria o separador mais indicado pois teríamos que evitar que as barras normais sejam tratadas como separadores, o que pode nos levar a uma linha complicada. Por exemplo, para substituir todas as ocorrências de `/etc/passwd` por `/var/lib/`:

```
\$ sed 's/\/etc\/passwd/\/var\/lib\//'
```

Um pouco confuso, não acha? Seria muito melhor neste caso usar um outro caractere como separador, para nos livrar da necessidade de escapar as barras com contra-barras. Usar um *underline*, por exemplo, resolveria o problema:

```
\$ sed 's_/etc/passwd_/var/lib/_'
```

Remover linhas inteiras do texto é muito simples, bastando usar a função `d`. Antes de passarmos para o próximo comando, vale a pena dar uma olhada em alguns detalhes importantíssimos do **sed**:

O sed diferencia maiúsculas de minúsculas.

Existem duas maneiras de fazer com que o **sed** passe a ignorar a diferença

(tornar-se *case-insensitive*), o primeiro é indicando todas as possibilidades, como no seguinte exemplo:

```
\$ cat texto.txt | sed '/[Ee][Rr][Rr][Oo]/d'
```

Isso removeria todas as linhas contendo a palavra `erro`, escrita de qualquer forma (Erro, ERRO, eRRo, etc). Uma outra solução, muito mais prática, é usar a *label* `I`, que diz ao **sed** que ele não deve diferenciar maíusculas de minúsculas ao tentar casar o padrão com o que ele encontrar no texto de entrada. A linha do exemplo anterior ficaria assim:

```
\$ cat texto.txt | sed '/erro/Id'
```

O que é muito mais simples, fácil e elegante.

Cuidados ao usar o sed:

Ao usar o **sed** no terminal, utilizar aspas simples envolvendo o comando é opcional, mas para evitar qualquer conflito com o shell, é recomendado utilizá-las sempre. Se não usarmos aspas simples, alguns caracteres como **$**,*,**\ e **!** serão tratados como metacaracteres do shell antes mesmo de serem enviados ao **sed**, o que fará com que o comando gere resultados estranhos ou nem seja executado.

Se quisermos que algo seja interpretado no meio do comando, como uma variável, por exemplo, devemos envolver o comando com aspas duplas:

```
\$ cat texto.txt | sed "s/abobrinha/\$CONTEUDO/"
```

A seguir iremos ver uma terceira alternativa para resolver esse tipo de conflito:

Os comandos do sed podem ser colocados em arquivo

É possível criar um arquivo executável com os comandos do **sed** e ao chamarmos o **sed** na linha de comando, ao invés de escrever tudo na hora, basta especificar o arquivo contendo os comandos através da opção `-f` do **sed**.

Como o **sed** é um comando complexo, cheio de opções e funções, merece um tutorial só para ele, não será possível cobrir nesta apostila tudo que o **sed** é capaz de fazer. Porém, se você quiser obter maiores informações sobre este comando, vale a pena dar uma olhada na documentação disponível. Além, é claro, de conferir a referência presente nesta apostila.

AWK

Assim como o **sed**, o **awk** não é exatamente um comando e está mais para uma linguagem de programação e a forma de processamento é idêntica, ou seja, é feita linha a linha. Esta linguagem é especialmente útil para extrair dados de arquivos organizados de forma sistemática em colunas com um separador fixo que não aparece no conteúdo do arquivo, um exemplo desse formato são os arquivos `/etc/group` e `/etc/passwd`. Note que as linhas dos dois arquivos são compostas por conteúdo separado por sinais de igual, e o sinal de igual só aparece como separador. Esse formato de arquivo é o ideal para se trabalhar com **awk**.

Da mesma forma que o **sed**, é possível escrever os comandos **awk** em um arquivo separado, usualmente com extensão `.awk`, e então chamar o interpretador **awk** da seguinte forma:

```
$ awk -f=comandos.awk [arquivo_de_entrada_1] ...
```

É possível também utilizar o **awk** direto da linha de comando, sempre utilizando a seguinte sintaxe:

```
$ awk 'condiçãofunção' [arquivo_de_entrada_1] ...
```

Onde condição pode ser literalmente uma condição (uma comparação), um padrão a ser encontrado nas linhas dos arquivos de entrada ou ainda uma das condições especiais do **awk** como `BEGIN` e `END` que servem para que uma ou mais funções sejam executadas no início ou logo após o término do tratamento dos arquivos de entrada, respectivamente. Se nenhuma condição for especificada as funções serão executadas para todas as linhas do arquivo de entrada.

Com o **awk** nós podemos utilizar variáveis, o que permite realizar diversos processamentos que não eram tão simples com os comandos vistos até agora. Vamos ver um exemplo simples usando as condições `BEGIN`, `END` e variáveis, além de operadores aritméticos.

Como seria o comando **awk** para simular o funcionamento do comando **wc -l** (comando 5.10, p. 76)?

Para começar, precisaremos de uma variável para armazenar o número de linhas conforme andamos pelo arquivo de entrada. E precisaremos inicializá-la com o valor 0 no início da execuçao do nosso comando, para isso precisaremos começar com a seguinte linha:

```
\$ awk 'BEGIN { numero_de_linhas = 0 } \
```

Note que a contrabarra ao final da linha serve apenas para escapar a mudança de linha, e isso só é necessário ao utilizarmos o **awk** diretamente na linha de comando ou embutido em um *shell script*. Agora que temos nossa variável criada e devidamente inicializada, devemos incrementá-la a cada linha encontrada. Para isso, temos o seguinte continuação do comando:

```
> { numero_de_linhas++ }\
```

E, finalmente, ao terminar de passar pelo arquivo de entrada precisamos mostrar na tela o total de linhas que encontramos:

```
> END{ print numero_de_linhas}' arquivo_de_entrada
```

Além de podermos usar variáveis à vontade no **awk** ele já nos oferece algumas específicas que são extremamente úteis, como a `NF`, que guarda o número do último campo da linha que está sendo processada, ou ainda a `NR` que guarda o número de linhas tratadas até o momento. Utilizando essas variáveis do **awk** o exemplo anterior se resume a isto:

```
\$ awk 'END{ print NR }' arquivo_de_entrada
```

Uma aplicação interessante da função print do **awk** é a possibilidade de imprimir um certo campo da linha que está sendo processada (similar ao **cut**) (comando 5.8, p. 75). Para fazer isso basta especificar o campo com um $ na frente:

```
\$ awk '{ print \$2 }' entrada
```

Essa linha de comando imprime todos os segundos campos de todas as linhas do

arquivo `entrada`. O separador de campos, por padrão é um espaço, mas é possível modificar o separador. O **awk** utiliza uma variável para guardar o separador de campos, chamada `FS`. Para mudar o separador de campos basta atribuir um novo valor a essa variável:

```
\$ awk 'BEGIN{ FS = ","}'
```

Isso faria com que a vírgula fosse usada como separador de campos ao invés do espaço. Isso mostra como o **awk** é mais versátil que o **cut** e outras ferramentas. O separador de campos pode mudar conforme os arquivos de entrada são analisados, e é possível utilizar uma variável para especificar qual campo se deseja imprimir.

Com o que vimos até aqui e com o conteúdo extra que pode ser encontrado na referência deste apostila podemos criar diversos comandos **awk** para vários fins, como calcular uma média entre vários valores armazenados em um arquivo, imprimir somente algumas linhas de um arquivo efetuando qualquer comparação de texto ou aritmética, entre outros.

Essa apostila vai se limitar ao básico do **awk** sem entrar em detalhes de programação avançada com **awk**, para saber mais sobre o programa existe uma extensa documentação disponível.

Exercício Resolvido 5.1 *Procurar no arquivo*

```
/pastas\_exercicios/Diversos/aprovadosUFPR.txt
```

todas as pessoas que tenham o sobrenome Teixeira. Fazer dois novos arquivos chamados Teixeiras_aleatorios.txt e Teixeiras_ordenados, apenas com o primeiro nome dessas pessoas. No primeiro arquivo os nomes devem estar arranjados de maneira aleatória e no segundo em ordem alfabética. Comparar quantos nomes iguais permaneceram na mesma linha, sendo o arquivo ordenado ou não.

```
$ cat aprovadosUFPR.txt | egrep TEIXEIRA | tr -s  | \
> cut -d -f2 | sort -R | tee Teixeiras_aleatorios.txt | \
> sort | tee Teixeiras_ordenados.txt
$ diff -y Teixeiras_aleatorios.txt Teixeiras_ordenados.txt | \
> grep -v '[<, >, |]'| wc -l
```

*Com apenas duas linhas de comando, podemos resolver esse problema. Na primeira, pesquisamos pelos nomes e formamos os arquivos pedidos. Na segunda linha, comparamos os dois arquivos. Como o comando **diff**, com o parametro -y, mostra quais linhas mudaram ou permaneceram iguais com os sinais ¡,¿ e —, usamos uma expressao regular para excluir essas linhas.*

Exercício Resolvido 5.2 *Você está responsável por entrar em contato com todos de sua empresa que tenham entre suas funções a de "analista pleno". Tudo que você tem em mãos é uma lista muito confusa contendo os nomes, e-mails e atribuições de cada funcionário (arquivo `empresa/tabelaEmpresa.txt`.*

É possível filtrar e melhorar a formatação das informações desta lista utilizando apenas o comando sed:

```
$ sed 's_&_\n_g; \                                    (1)
>/.*analista pleno.*/Ip' \                            (2)
>-n tabelaEmpresa.txt                                 (3)
```

Para imprimir uma lista contendo apenas o contato dos Analistas Plenos com uma formatação mais legível, primeiro vamos trocar os & responsáveis pela separação

das informações por quebras de linha (1). Desta forma temos uma informação por linha, facilitando a leitura.

Após isso devemos filtrar os campos, para que apenas os registros dos Analistas Plenos sejam impressos. O uso de uma expressão regular juntamente com a função ***p*** *do* ***sed*** *em (2) resolve este problema.*

Para finalizar, em (3) especificamos a opção ***-n*** *para o* ***sed*** *pois sem ela ele passaria o arquivo todo para a saída padrão, inutilizando nossa busca, e passamos o arquivo com os dados (*`tabelaEmpresa.txti`*) como parâmetro. Este comando pode ser escrito em uma única linha.*

Comando 5.8 `cut` Remove seções de cada linha de arquivos.

NOME

cut

SINTAXE

`cut` [opcões] [arquivo] ...

DESCRIÇÃO

O comando **`cut`** imprime apenas as partes selecionadas das linhas de um arquivo para a saída padrão, sendo possível selecionar campos, caracteres ou bytes.
As LISTAS que servem para selecionar as partes desejadas funcionam da seguinte forma:

`N` Seleciona apenas a parte N;

`-N` Seleciona do começo da linha até N;

`N-M` Seleciona de N até M;

`N-` Seleciona de N até o fim da linha.

As listas podem ser combinadas através de vírgulas. Por exemplo, o comando a seguir tem como DELIMITADOR o caractere : (dois pontos) e seleciona as colunas 1, 2, 3, 4, 6, 18, 19 e 20:

```
$ cut -d ":" -f 1-4,6,18-20 texto.txt
```

OPÇÕES

`-s` Não imprime nenhuma linha que não contenha nenhum delimitador de campo;

`-b LISTA`
Seleciona apenas estes bytes;

`-c LISTA`
Seleciona apenas estes caracteres;

`-d DELIMITADOR`
Utiliza DELIMITADOR ao invés de TAB como separador de campo;

`-f LISTA`
Seleciona apenas estes campos. Imprime qualquer linha que não contenha nenhum DELIMITADOR a não ser que a opção -s seja especificada.

Comando 5.9 `tr` Substitui ou deleta caracteres.

NOME

tr

SINTAXE

tr [opcões] Conjunto 1 [Conjunto 2]

DESCRIÇÃO

O comando **tr** realiza substituição ou remoção de caracteres da entrada padrão, produzindo seu resultado na saída padrão. Para realizar substituição de caracteres nenhum parâmetro deve ser específicado e os dois conjuntos de caracteres devem estar presentes.

OPÇÕES

`-d` Deleta os caracteres do Conjunto 1;

`-s` Substitui cada sequência de dois ou mais caracteres do Conjunto 1 por uma única ocorrência do caractere, removendo repetições.

Comando 5.10 `wc` Conta o número de linhas, bytes, caracteres ou palavras.

NOME

wc

SINTAXE

wc [opcões] [arquivo] ...

DESCRIÇÃO

O comando wc serve para contar o número de linhas, bytes, caracteres ou palavras de um arquivo.

OPÇÕES

`-w` Imprime a contagem de palavras;

`-l` Imprime a contagem de linhas;

`-c` Imprime a contagem de bytes;

`-m` Imprime a contagem de caracteres;

`-L` Imprime o comprimento da linha mais longa.

Comando 5.11 `nl` Numera linhas.

NOME

nl

SINTAXE

nl [arquivo] ...

DESCRIÇÃO

Mostra o conteúdo do arquivo na saída padrão com a numeração das linhas ao lado de cada linha. Se nenhum arquivo for indicado, lê da entrada padrão.

Comando 5.12 head Imprime as primeiras linhas de um arquivo.

NOME

head

SINTAXE

head [opções] [arquivo] ...

DESCRIÇÃO

Mostra as 10 primeiras linhas do arquivo na saída padrão, a não ser que alguma opção seja utilizada. Se nenhum arquivo for especificado, lê da entrada padrão.

OPÇÕES

`-v` Imprime o nome do arquivo antes de qualquer outra saída;

`-n [-]NUM`
Imprime as NUM primeiras linhas do arquivo na saída padrão ou, se o - for utilizado diante de NUM, imprime todas menos as NUM últimas linhas na saída padrão;

`-c [-]NUM`
Imprime os NUM primeiros bytes do arquivo na saída padrão ou, se o - for utilizado diante de NUM, imprime todos menos os NUM últimos bytes na saída padrão.

Comando 5.13 `tail` Imprime as últimas linhas de um arquivo.

NOME

tail

SINTAXE

tail [opções] [arquivo] ...

DESCRIÇÃO

Mostra as 10 últimas linhas do arquivo na saída padrão, a não ser que alguma opção seja utilizada. Se nenhum arquivo for especificado, lê da entrada padrão.

OPÇÕES

`-v` Imprime o nome do arquivo antes de qualquer outra saída;

`-n [+]NUM`
Imprime as NUM últimas linhas do arquivo na saída padrão ou, se o + for utilizado diante de NUM, imprime todas menos as NUM primeiras linhas na saída padrão;

`-c [+]NUM`
Imprime os NUM últimos bytes do arquivo na saída padrão ou, se o + for utilizado diante de NUM, imprime todos menos os NUM primeiros bytes na saída padrão.

Comando 5.14 `sort` Ordena arquivos por linha.

NOME

sort

SINTAXE

`sort [opções] [arquivo] ...`

DESCRIÇÃO

Ordena as linhas de um arquivo numérica ou alfabeticamente e mostra o resultado na saída padrão. Se nenhum arquivo for especificado, lê da entrada padrão.

OPÇÕES

`-d` Ordena por ordem alfabética considerando apenas caractéres alfanuméricos;

`-f` Desconsidera diferenças entre maíusculas e minúsculas;

`-n` Ordena numericamente;

`-h` Considera números em formato legível para humanos (ex: 20k, 1G, etc.);

`-R` “Ordenação” aleatória;

`-r` Inverte os resultados da ordenação.

Comando 5.15 bc Calculadora.

NOME

bc

SINTAXE

bc [-l] expressão

DESCRIÇÃO

Permite a realização de operações diretamente na linha de comando, mostrando o resultado da expressão na saída padrão.

OPÇÕES

`-l` Carrega a biblioteca matemática padrão.

PRINCIPAIS EXPRESSÕES

`expr1 + expr2`
Executa a operação de soma entre expr1 e expr2. Ex: $2 + 2$;

`expr1 - expr2`
Executa a operação de subtração entre expr1 e expr2. Ex: $10 - 2$;

`expr1 / expr2`
Executa a operação de divisão entre expr1 e expr2. Ex: $\frac{143}{5}$;

`expr1 * expr2`
Executa a operação de multiplicação entre expr1 e expr2. Ex: $5 * 4$;

`expr1 ^ expr2`
Executa a operação de potenciação entre expr1 e expr2. Ex: 2^5;

`expr1 % expr2`
Executa a operação de módulo (o resultado é o resto da divisão) entre expr1 e expr2. Ex: 4%2.

Comando 5.16 sed Edita fluxos de texto.

NOME

sed

SINTAXE

sed [opções] funções [arquivo] ...

DESCRIÇÃO

Comando usado para editar fluxos de texto de forma automática e eficiente. Se nenhum arquivo for especificado, lê o conteúdo da entrada padrão. As funções do sed têm uma sintaxe específica e não muito intuitiva. Para uma explicação detalhada ver a seção *Manipulação avançada de texto* (ver 5.3.6).

OPÇÕES

`-n` Desabilita a impressão automática na saída padrão;

`-e função`
Adiciona a função à lista de comandos a serem executados;

`-f arquivo`
Ao invés de escrever os comandos do sed na linha de comando, estes estarão armazenados no arquivo.

RESUMO DAS PRINCIPAIS FUNÇÕES

`d` Deletar linha(s);

`/padrão/`
Endereçar linhas que contenham o padrão (expressões regulares são suportadas);

`s` Busca e substituição de padrão: 's/padrãoBuscado/novoConteúdo' (percorre cada linha em busca do "padrãoBuscado" e caso encontre realiza sua substituição por "novoConteúdo";

`p` Imprimir linha(s);

`=` Imprimir número da linha atual;

`q` Sair do sed.

Comando 5.17 `awk` Filtra e edita arquivos.

NOME

awk

SINTAXE

`awk` [opções] ['texto do programa'] [arquivo] ...

DESCRIÇÃO

Executa busca por padrões, filtragem por campos, operações aritméticas, oferece uso de variáveis, laços e operações condicionais na hora de manipular um arquivo de texto especificado como parâmetro ou, caso nenhum arquivo seja indicado, manipular o conteúdo lido da entrada padrão.

OPÇÕES

`-f arquivo`
: Ao invés de escrever o ['texto do programa'] diretamente ao chamar o awk, os comandos serão lidos do arquivo;

`-F valor`
: O caractere separador de campos (FS) passa a ser valor;

`-v var=valor`
: A variável do programa var recebe o valor valor.

5.4 Compactação e Afins

Compactar é uma forma de representar dados de forma eficiente, ou seja, fazer com que vários arquivos sejam representados por um único arquivo de tamanho reduzido. Se um bloco de dados é compactado usando uma determinada extensão, este só pode ser descompactado por um software que trate esta mesma extensão. No Linux existe uma grande variedade de extensões de arquivos compactados. Entre as mais utilizadas estão: bz2, gz e zip. Nesta serão serão abordados apenas os comandos gzip e gunzip, que lidam com a extensão .gz, porém existem comandos muito similares que lidam com outras extensões, como zip, unzip, rar, unrar, bzip e bunzip.

Já a operação de split consiste em dividir um arquivo em pequenas partes, sem mudança de representação ou do espaço total ocupado por este. Ele apenas divide o conteúdo do arquivo em pedaços e distribui estas partes em diversos arquivos. Para juntar as partes geradas pelo split pode-se utilizar o comando cat (comando 4.1, p. 36), redirecionando a saída das partes para um outro arquivo, gerando o arquivo original.

Já o conceito de armazenamento, ou empacotamento, é o inverso do split, ao invés de dividir um arquivo em várias partes, junta-se vários arquivos em apenas um novo arquivo, porém sem compactá-lo.

Exercício Resolvido 5.3 *Como dividir um arquivo em partes de 500kb?*

```
$ split -b500 nome_arquivo nome_da_parte
```

O comando ***split*** *é o responsável pela divisão se arquivos, a opção -b500 indica o tamanho de cada parte, nesse caso 500kb, e o nome dos arquivos resultantes será parte_a,parte_b,partec, . . .*

Exercício Resolvido 5.4 *Como gerar um arquivo que una todos os diretórios abaixo* `/pasta_exercicios` *?*

```
$ cd /home/user $ tar -cf diretorios.tar pasta_exercicios
```

O comando ***tar*** *é o responsável pela concatenação, a opção -cf criam um novo tar e especifica o tar a ser usado. Após as opções indica-se o diretório a ser concatenado.*

Comando 5.18 `split` Divide um arquivo em partes menores.

NOME

split

SINTAXE

`split` [opcões] [arquivo [prefixo]]

DESCRIÇÃO

Este comando permite dividir um arquivo em várias partes. Por padrão, ele dividirá um arquivo a cada 1000 linhas, porém isso pode ser alterado com suas opções. O prefixo padrão é 'x', logo, os arquivos de saídas são denominados xaa, xab...

OPÇÕES

`-b` TAM

Divide o arquivo pelo tamanho desejado em bytes. Por exemplo, -b512 irá colocar 512 bytes em cada arquivo de saída. Pode-se também usufruir das letras k, m e g para representar kilobytes, megabytes e gigabytes, respectivamente. Por exemplo, -b500m divide em partes de 500 megabytes;

`-l` TAM

Divide o arquivo pelo número de linhas. Por exemplo, -l650 irá colocar 650 linhas em cada arquivo de saída.

Comando 5.19 `tar` Empacota arquivos.

NOME

tar

SINTAXE

`tar` [opcões] [-f arquivo destinatário(.tar)] [-C diretório] [arquivo 1] ...

DESCRIÇÃO

O comando **`tar`** basicamente empacota e desempacota arquivos. Porém com algumas opções, este comando serve também para compactar o pacote gerado. Por exemplo, para criar um arquivo chamado `backup.tar` que contenha os arquivos `1.txt` e `2.txt` da pasta corrente, o `arquivo 3.avi` que está na pasta `Videos` e a pasta `Musicas` inteira:

```
$ tar -cf backup.tar 1.txt 2.txt Musicas/ -C Videos/ 3.avi
```

O uso mais comum deste comando:

```
$ tar -czvf backup.tar.gz Home/
```

E para descompactar:

```
$ tar -xzvf backup.tar.gz
```

Ou descompactar na pasta raiz:

```
$ tar -xzvf backup.tar.gz -C /
```

OPÇÕES

`-f` Este é mais que uma opção, é item quase que obrigatório, pois especifica o arquivo tar a ser usado;

`-c` Cria um novo arquivo tar;

`-t` Exibe o conteúdo de um arquivo tar;

`-x` Extrai arquivos de um arquivo tar;

`-C` Especifica o diretório dos arquivos a serem armazenados;

`-z` Comprime ou extrai arquivos tar resultante com o gzip;

`-j` Comprime ou extrai arquivos tar resultante com o bz2;

`-v` Exibe detalhes da operação;

`-w` Pede confirmação antes de cada ação;

`-r` Acrescenta arquivos a um arquivo tar;

`-p` Mantém as permissões originais dos arquivos.

Comando 5.20 `gzip` Compacta um arquivo.

NOME

gzip

SINTAXE

`gzip` [opcões] arquivo

DESCRIÇÃO

Gzip é a abreviação de GNU zip, um Software Livre de compactação de dados. A extensão gerada pelo gzip é o .gz, e seu formato contém apenas um arquivo comprimido. É comum gerar um arquivo contendo diversos outros arquivos com o programa tar, e depois comprimi-lo com o gzip, gerando um arquivo .tar.gz. Após compactar um arquivo e gerar o .gz, o arquivo original é excluído. O comando a seguir gera o arquivo arquivo.txt.gz.

```
$gzip arquivo.txt
```

OPÇÕES

`-c` Mostra conteúdo do arquivo a ser compactado na saída padrão;

`-d` Descompacta um arquivo .gz.

NOTAS

É importante ressaltar que este comando exclui o arquivo original. Porém com a opção -c é possível redirecionar o conteúdo para outro arquivo, mantendo as informações originais. Assim:

```
$ gzip -c arquivo.txt > NovoArquivo.txt
```

Comando 5.21 `gunzip` Descompacta um arquivo.

NOME

gunzip

SINTAXE

`gunzip` arquivo

DESCRIÇÃO

Serve para descompactar um arquivo .gzip.

```
$gunzip arquivo.gz
```

5.5 Expandindo Meu Território

Uma das utilizações e necessidades mais comuns na computação moderna é a possibilidade de transferência de dados, comunicação e acesso remoto à informações e dados. Essas aplicações são comumente utilizadas, porém a grande maioria das vezes em modo gráfico. Veremos aqui que estas funcionalidades podem ser executadas através de linhas de comando, rapidamente e sem complicações.

É importante saber a sintaxe quando se referência um computador remotamente ou um determinado arquivo neste computador.

usuário@hostname:endereço_do_arquivo

Nesta sintaxe, usuário refere-se ao usuário da máquina a ser acessada remotamente, esta máquina é referenciada através de seu hostname. Caso o acesso seja a um determinado arquivo ou diretório, deve-se colocar : (dois pontos) e após isto o endereço completo onde encontra-se o arquivo desejado.

Acesso Remoto

A necessidade de acessar um outro computador para utilizar ou ter acesso a seus dados é muito comum, por isso existem protocolos de acesso remoto. O protocolo SSH (**S**ecure **SH**ell) permite a conexão com outro computador, permitindo executar comandos em uma unidade remota, com a vantagem da conexão entre o cliente e o servidor ser criptografada.

Web

É extremamente comum utilizar a internet com navegadores em modo gráfico. Também é possível navegar via linhas de comando, além de realizar *download* de um modo muito prático.

Exercício Resolvido 5.5 *Como acessar remotamente uma máquina, com disponibilidade do acesso gráfico?*

```
$ssh -X user@ip
```

O comando ***ssh*** *é o responsável pelo acesso remoto, e a opção -X permite a visualização do modo gráfico.*

Comando 5.22 `ssh` Acesso remoto.

NOME

ssh

SINTAXE

`ssh` [**`-CX`**] usuário@hostname [comando]

DESCRIÇÃO

Comando utilizado para acesso remoto utilizando o protocolo SSH. Caso o comando não seja especificado a conexão será de acesso ao terminal, como se estivesse com um terminal aberto e com o usuário em frente à máquina. Para encerrar a conexão, pode-se utilizar o comando **`exit`**.

OPCOES

`-C` gera uma transmissão com dados compactados;

`-X` gera uma transmissão que suporta a parte gráfica.

Comando 5.23 `scp` Tranferência remota.

NOME

scp

SINTAXE

`scp` [**`-lP`**] [usuário@hostname:]endereço_origem [usuário@hostname:]endereço_destino

DESCRIÇÃO

Comando utilizado para copiar arquivos remotamente. Pode-se copiar arquivos de uma máquina remota para a outra, sem necessidade de copiá-los para a máquina que está sendo utilizada fisicamente.
Para fazer referência à máquina local é possível apenas suprimir seu usuário@hostname:

OPCOES

`-l` limita a velocidade da transferência;

`-P` indica uma porta específica.

Comando 5.24 `sshfs` Montar pastas remotamente

NOME

sshfs

SINTAXE

`sshfs` `usuário@hostname:diretório ponto_de_montagem`

DESCRIÇÃO

O comando sshfs é utilizado para montar pastas remotamente. Estas pastas ficam sincronizadas, ou seja, qualquer alteração em uma afetará a outra.
O ponto_de_montagem deve ser uma pasta vazia que assumirá o papel do diretório sincronizado. Por exemplo, pode-se criar uma pasta `remote` no seu `Desktop` para servir para isso. O que resultaria em uma linha de código assim:

```
$ sshfs usuário@hostname:Desktop ~/Desktop/remote
$ sshfs usuário@hostname:/home/usuário/Desktop ~/Desktop/remote
```

Para desmontar o diretório montado remotamente pode-se usar o comando **`fusermount`** desta maneira:
`fusermount -u` ponto_de_montagem
Para o exemplo citado acima:

```
$ fusermount -u remote
```

Comando 5.25 `wget` Baixa arquivos de um servidor.

NOME

wget

SINTAXE

`wget` `[URL] ...`

DESCRIÇÃO

O comando **`wget`** recebe como parametro uma ou mais URLs e baixa o arquivo para o computador.

Comando 5.26 `wget` Visualiza uma página web no modo texto.

NOME

lynx

SINTAXE

`lynx` `URL`

DESCRIÇÃO

O comando lynx é na verdade um navegador em modo texto. Não se iluda, por ser em modo texto, a navegação não será como você deve estar acostumado. Pois o lynx não exibe imagens nem distingue fontes porém, isso deixa a navegação extremamente rápida para quem se habitua a este comando.

5.6 Exercícios

Para resolver esses desafios, utilize as man pages, juntamento com as informações da apostila. Não esqueça de fazer o download do diretório de exercicios que acompanha a apostila.

Exercício 5.6 *Procure na sua home por todos os arquivos que só possuem permissão de leitura.*

Exercício 5.7 *Caso o seu computador desligue enquanto você faz um trabalho que foi salvo algumas vezes, e não se tem certeza do nome nem de onde estão os arquivos. Como encontrar todos arquivos acessados a menos de 30 minutos e que tenham extensão .txt?*

Exercício 5.8 *Para liberar espaco no HD, basta deletar alguns arquivos grandes que sejam inúteis. O que deve-se fazer para listar todos os arquivos com mais de 100 Mb e que possuem permissão para que o usuário atual os exclua? rabalho.*

Exercício 5.9 *Existe um arquivo em uma servidor remoto cujo conteúdo lhe interessa, porém você não sabe em que diretório se encontra este arquivo. Como copiar o conteúdo desse arquivo para a sua máquina usando o menor número de linhas de comando possíveis?*

Exercício 5.10 *Vários arquivos da sua home precisam ser transferidos para unidades de armazenamento externos, que são em quantidade limitada, e em cada uma dessas unidades tem pouco espaço disponível. Como transferir estes arquivos para as unidades? Existe mais de uma forma de executar esta ação? Se sim, qual a maneira mais eficiente de se resolver este problema?*

Exercício 5.11 *Você está cursando o segundo período do curso de Ciência da Computação, e para se familiarizar com a linguagem de programação C, decidiu escrever um pequeno programa para simular o famoso Jogo da Vida (`/Programacao/jogoDaVida.c`). Depois de algum tempo escrevendo código e alguns testes, você percebe que não deveria ter usado o tipo* int *em suas declarações de variáveis, e sim* unsigned int*. Para não perder tempo procurando por cada ocorrência, escreva uma linha de comando utilizando o* **`sed`** *que automatize esta tarefa.*

Exercício 5.12 *Você tem uma lista com nomes, telefones, e-emails e endereços de colegas `/empresa/enderecos.txt`. Você quer saber quantos destes contatos utilizam o serviço de e-mail do departamento de informática da UFPR (@inf.ufpr.br) e quer saber o nome e telefone destas pessoas. Como você faria isso utilizando o comando* **`awk`***?*

Apêndice A

Instalação do Ubuntu

A.1 Obtendo o Ubuntu

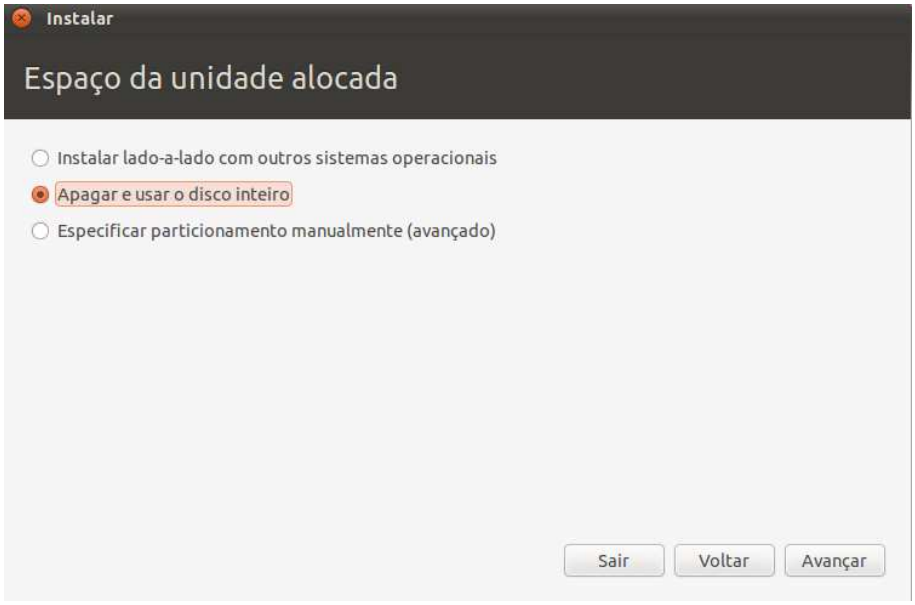
No site do ubuntu (http://www.ubuntu.com/) tem o sistema operacional para baixar gratuitamente e também se explica de forma fácil e clara como gravar um cd ou um pendrive bootável.

A.2 Instalação

Após baixar e fazer o cd ou pendrive bootável, reinicie o computador e configure a bios para bootar pelo pendrive ou cd. Se não ocorrer nenhum erro, o computador estará executando o Ubuntu e você poderá usar o sistema como quiser sem nenhum problema. Então para instalar o Ubuntu haverá no desktop um icone, que inicializará o processo de instalação. A instalação é fácil, em grande parte só pedirá algumas informações, mas o particionamento do disco rígido é um pouco difícil e há o risco de se perder os dados do disco.

A.2.1 Particionamento

O disco rígido no computador pode ser dividido em várias partes. Cada parte é chamada de partição, nela deverá ter apenas um sistema operacional e será tratada como se fosse um outro disco rígido. Então para instalar o Ubuntu é necessario criar uma partição apenas para o Ubuntu, de no mínimo 8gb, e no sistema de arquivos do linux (Ext3, Ext4 ou ReiserFS).



Caso 1: Instalar lado-a-lado com outro Sistema Operacional

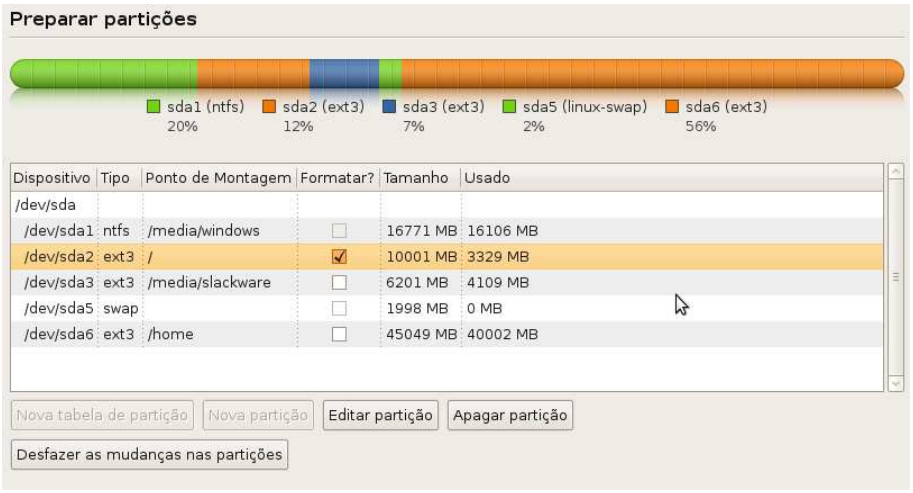
A opção "Instalar o lado a lado" irá pegar o espaço vazio da partição e criará uma nova partição para o Ubuntu, isso sem apagar os dados, mas levará bastante tempo. Caso o computador for interrompido, pode-se perder os dados do disco rígido. Então recomenda-se, antes, salvar os dados mais importantes e desfragmentar o disco.

Caso 2: Apagar e usar o Disco Inteiro

A opção "usar o Disco Inteiro" simplesmente apagará todos os dados do disco e criará as partições, como o sistema achar melhor.

Caso 3: Especificar particionamento manualmente

Na opção "Particionamento manual", você irá configurar as partições do disco como você quiser. Por exemplo:



`Dispositivo`
: Indica os discos rígidos e suas partições, por exemplo, sda1: disco A e partição 1;

`Tipo` Sistema de arquivo utilizado ou que será colocado na partição;

`Ponto de Montagem`
: O diretório onde será montado a partição. Portanto é obrigatório indicar uma partição para a raiz /, pois é onde será instalado o sistema;

`Formatar?`
: Se selecionado, então a partição terá todos os dados apagados, mas é a única forma de alterar o sistema de arquivo de uma partição;

`Tamanho`
: Tamanho total da partição;

Entendido as informações da ferramenta, agora falta particionar o disco para o Ubuntu. Normalmente se utiliza uma partição de no mínimo de 8gb para a raiz, outra de no máximo 2gb para Swap, que apenas serve para dados temporários do sistema, ou seja, o usuário não tem acesso; e outra de tamanho livre para /home, onde ficará os dados dos usuários do sistema, mas essa partição não é obrigatória. Porém, recomenda-se utilizar uma partição separada para o /home, pois caso necessite formatar a raiz em outra situação, os dados do usuarios não serão apagados.
Após particionar, o resto da instalação será mamão com açucar.

Apêndice B

Instalação de Programas

B.1 Apt-Get

A instalação de novos programas usando os comandos Apt-Get ou Aptitude são bastante faceis de usar, porém o inconveniente é ficar limitado aos programas contidos no repositório. O funcionamento desse sistema é bem simples, em algum lugar do mundo existe um servidor onde fica guardado vários pacotes e programas gratuitos. Então em seu computador existe um arquivo, que armazena o endereço desse servidor e quais pacotes e programas o repositório contém.

Comando B.1 `apt-get` Instala algum pacote ou programa do repositório.

NOME

apt-get

SINTAXE

`apt-get` `[opcões]` `programa`

DESCRIÇÃO

O comando apt-get gerencia os pacotes e programas do computador, pode-se instalar ou remover programas do sistema. Como a instalação de programas envolve sempre arquivos do sistema, é necessario ser root ou utilizar junto o comando sudo.

OPÇÕES

`update`
: Atualiza a lista de pacotes e programas;

`install`
: Baixa e instala algum pacote ou programa;

`autoremove`
: Remove algum pacote ou programa já instalado.

B.2 Pacotes Deb

O sistema operacional Debian desenvolveu um sistema para facilitar a instalação de pacotes que é utilizado por grande parte dos sistemas baseados no Debian, inclusive o Ubuntu. Grande parte desses pacotes estão no site do debian e após baixados no computador, podem-se ser instalados pelo comando dpkg.

Comando B.2 dpkg

NOME

dpkg

SINTAXE

dpkg [Opcões] Pacote

DESCRIÇÃO

O comando dpkg é a base do funcionamente do Apt-Get. Ele é capaz de instalar e gerenciar os pacotes Deb.

OPÇÕES

`-i` Instala um novo pacote .deb;

`-r` Remove um pacote controlado pelo repositorio;

`-l` Mostra os pacotes contidos no repositorio;

`-s` Mostra o status de algum pacote do repositorio no sistema;

`--unpack`
Descompacta um pacote .deb.

EXEMPLOS

dpkg -i pacote.deb dpkg -r pacote dpkg -l texlive* dpkg -s openssh-server

B.3 Na Unha

No Linux, quando você baixar um novo programa, normalmente vem apenas o codigo-fonte do programa e não o codigo executável como no Windows. Se tem o codigo executável, para instalar é apenas copiar os arquivos, jogar para alguma pasta e executar o programa. Porém o codigo executavel é dependente de arquitetura de computador e retira a liberdade de alterar o software. Então no linux para instalar um novo programa é necessario transformar o codigo-fonte em codigo executavel, essa transformação chama-se compilar um programa.
Para facilitar essa etapa de transformação, foi desenvolvido um método padrão. Todo programa que segue esse padrão, tem um script na pasta ... chamado configure. Então antes de compilar algum programa é necessario executar o script configure, que verificará se todas as bibliotecas utilizadas pelo programa já estão instaladas. Caso falte alguma biblioteca, tente instalar pelo apt-get, caso contrario a instalação tornará extramente trabalhoso e cansativo. Se der tudo certo, o script criará um arquivo na mesma pasta chamado de Makefile, então agora é só executar o comando make, que compilará o codigo-fonte. Aguarde um bom tempo, alguns programas demoram horas para terminar de compilar. Agora o programa tem o codigo-executavel e só falta copiar o arquivo executável para o sistema e configurar. Essa parte é feita pelo comando make install, lembrando que como a alteração é feita nos diretorios do sistema é necessario ser root ou utilizar o comando sudo junto nessa última etapa.

Resumindo
Descompacte o novo programa em algum lugar
Entre no diretorio, onde o programa foi descompactado
execute o arquivo configure
execute o comando make
execute o comando make install (sudo make install)

Apêndice C

Comandos de Rede

Comando C.1 `ifconfig` Mostra as conexoes de rede do computador

NOME

ifconfig

SINTAXE

`ifconfig` `[Opcoes]`

DESCRIÇÃO

O comando ifconfig mostra como esta configurado os dispositivos de rede.

OPÇÕES

`-a` Mostra todos os dispositivos de rede;

`up Dispositivo`
liga o dispositivo de rede;

`down Dispositivo`
desliga o dispositivo de rede;

EXEMPLOS

ifconfig

Saida:
Dispositivo Informaçoes do Dispositivo
eth0 Link encap:Ethernet HWaddr 00:19:d1:26:ed:c5 (Tipo e MAC)
inet addr:192.168.0.1 Bcast:192.168.0.255 Mask:255.255.255.0 (IP)
inet6 addr: fe80::219:d1ff:fe26:edc5/64 (IPv6)
... RX bytes:484457852 (484.4 MB) TX bytes:36230552 (36.2 MB)

eth1 Link encap:Ethernet HWaddr 00:e0:7d:a7:e7:59
inet addr:192.168.0.2 Bcast:192.168.0.255 Mask:255.255.255.0
inet6 addr: fe80::2e0:7dff:fea7:e759/64 Scope:Link
...
RX bytes:35093357 (35.0 MB) TX bytes:801545308 (801.5 MB)
Interrupt:21 Base address:0x1100

...

ifconfig down eth0

Comando C.2 `dhclient` conecta a rede local

NOME

dhclient

SINTAXE

`dhclient` `[Dispositivo]`

DESCRIÇÃO

O comando dhclient conecta ao servidor da rede local (Servidor DHCP) e pede para o servidor colocar o novo computador na rede. Após a execução do comando, se não houver nenhum erro, o computador já estará na rede local. Então pode-se acessar os outros computadores e acessar a internet, caso a rede tiver internet.

EXEMPLOS

dhclient
dhclient eth0

Comando C.3 `netcat` Conecta em algum servidor

NOME

nc

SINTAXE

`nc` `[Opcoes] Servidor Porta`

DESCRIÇÃO

OPÇÕES

`-v` Mostra o log, se ocorreu algum erro ou não;

EXEMPLOS

nc www.google.com.br 80
após conectado envie o seguinte dados ????, então você receberá a pagina do google em html

Comando C.4 `netcat` Cria um servidor

NOME

nc

SINTAXE

nc `-lp Porta`
nc `-lp Porta -e Comando`

DESCRIÇÃO

O comando netcat com a opção -l coloca o computador escutando a Porta selecionada e simplesmente mostra os dados recebidos na porta. Ou também, além de mostrar os dados, é possivel executar algum comando, quando receber algum dados na porta, isso é feito adicionando a opção -e.

OPÇÕES

`-lp Porta`
: Coloca o computador para escutar (-l) na porta selecionada (-p);

`-e Comando`
: Ao receber alguma informação, o computador executa o comando selecionado

EXEMPLOS

Cria um servidor escutando a porta 5000
nc -lp 5000

Cria um servidor escutando a porta 5000 com acesso ao terminal
nc -lp 5000 -e /bin/bash

Conecta ao servidor
nc Servidor 5000

Comando C.5 `netcat` Envia dados para um computador

NOME

nc

SINTAXE

nc `Servidor Porta Arquivo`

DESCRIÇÃO

OPÇÕES

`-v` Imprime o nome do arquivo antes de qualquer outra saída.

Comando C.6 `netcat` Realiza um scanner pelas portas

NOME

nc

SINTAXE

nc `-vz Servidor Portas -w Timeout`

DESCRIÇÃO

O comando netcat dessa forma, cria um scanner pelas portas de algum servidor e irá dizer quais portas estão abertas.

OPÇÕES

`-v` Mostra o log, se ocorreu algum erro ou não;

`-z` Conecta e logo desconecta do servidor, necessario pois se houver alguma porta aberta o nc irá se conectar e só sairá se o usuario desconectar;

`-w Timeout`
Tempo limite para conectar ao servidor;

EXEMPLOS

nc -vz www.google.com.br 80-90 -w 3

Comando C.7 `netcat` Cria um proxy

NOME

nc

SINTAXE

nc `-l Porta`

DESCRIÇÃO

OPÇÕES

`-v` Imprime o nome do arquivo antes de qualquer outra saída.

Apêndice D

Editores de Texto

Este capítulo ensinará como usar um editor de texto no terminal do linux, que pela sua grande quantidade de ferramentas disponiveis e portando apenas o teclado como forma de entrada de dados, teve o seu uso dificultado. Também é importante saber pelo menos o mínimo, pois pode haver alguma circustância, da qual você terá apenas o modo texto e precisará dele para fazer o que você quer.

D.1 Vi

D.1.1 Introdução

O Visual Interface (VI) foi criado no inicio da década de 80. Ele contém dois modos de operação: Modo de Comando e Modo de Inserção. O usuário só pode estar em apenas um modo, que pode ser trocado. Cada modo altera o significado das teclas, por exemplo, a tecla 'D' no modo de comando indica ao computador para deletar uma linha, já no modo de inserção, o computador insere a letra 'D' no texto. Quando entrar no editor, por padrão, ele estará no modo de comando.

D.1.2 Comandos

MC = Modo de Comando ME = Modo de Edição

MC - Entrar no Modo de Edição

`i` entra no modo de edição;

`a` entra no modo e o cursor avança um caracter;

`r` entra no modo e coloca o cursor no inicio da linha atual;

`A` entra no modo e coloca o cursor no final da linha atual;

`o` entra no modo, cria uma nova linha abaixo da atual e coloca o cursor na nova linha.

ME - Entrar no Modo de Comando

ESC entra no modo de comando;

MC - Arquivo

`:e NomeArquivo`
abre o arquivo NomeArquivo;

`:w` salva o arquivo atual

`:w NomeArquivo`
salva o arquivo corrente no arquivo NomeArquivo;

`:w! NomeArquivo`
força salvar o arquivo corrente no arquivo NomeArquivo, use esse caso ocorra algum erro no :w;

`:q` sai sem salvar, porem ele nao sai do programa se o arquivo foi modificado e nao salvado;

`:q!` força sair sem salvar, caso o arquivo foi modificado;

`:wq` salva e sai;

MC - Editar

`d` recorta o texto a partir do cursor até o final da linha;

`dw` recorta a palavra sobre o cursor;

`dd` recorta a linha corrente;

`y$` copia o texto a partir do cursor até o final da linha;

`yw` copia a palavra sobre o cursor;

`yy` copia a linha corrente;

`p` cola depois do cursor;

`P` cola antes do cursor;

`x` deleta o caracter sobre o cursor;

`dw` deleta a palavra sobre o cursor;

`dd` deleta a linha atual;

`D` deleta a linha a partir da posição do cursor ate o final;

`N,M d` apaga da linha N até a M

`u` desfaz a última modificação;

`U` desfaz todas as modificações feitas na linha atual;

MC - Busca e substituição

`/exp. regular`
procura exp. regular no texto;

`n` vai para a próxima ocorrência;

`N` vai para a ocorrência anterior;

`:s/exp. regular/string/`
substitui a primeira ocorrência de exp. regular por string.

`:s/exp. regular/string/g`
substitui todas as ocorrências de exp. regular por string na linha atual.

`:L1,L2 s/exp. regular/string/g`
substitui todas as ocorrências de exp. regular por string da linha L1 até L2.

`:% s/exp. regular/string/g`
substitui todas as ocorrências de exp. regular por string em todo o texto.

MC - Navegação

`Ctrl` + `f`
move o cursor para proxima página;

`Ctrl` + `b`
move o cursor para página anterior;

`H` move o cursor para a primeira linha da página;

`L` move o cursor para a ultima linha da página;

`h` move o cursor um caracter a esquerda;

`j` move o cursor para a proxima linha;

`k` move o cursor para linha anterior;

`l` move o cursor um caracter a direita;

`w` move o cursor para o inicio da próxima palavra, ignorando a pontuacão;

`W` move o cursor para o inicio da próxima palavra, não ignorando a pontuacão;

`b` move o cursor para o inicio da palavra anterior, ignorando a pontuacão;

`B` move o cursor para o inicio da palavra anterior, não ignorando a pontuacão;

`0` move o cursor para o inicio da linha atual;

`$` move o cursor para o final da linha atual;

`nG` move o cursor para a linha n;

`G` move o cursor para a última linha do arquivo;

ME - Edição de Texto

`Teclas de A-Z`
insere no texto o caracter correspondente

D.2 Emacs

D.2.1 Introdução

O Emacs é outro editor de texto bastante utilizado. Ele facilita um pouco o uso, pois não usa estados como o VI e sim combinações de teclas.

D.2.2 Comandos

Arquivo

`Ctrl` + `x` e `Ctrl` + `f` `NomeArquivo`
abre o arquivo NomeArquivo;

`Ctrl` + `x` e `Ctrl` + `s` `NomeArquivo`
salva o arquivo corrente no arquivo NomeArquivo;

`Ctrl` + `x` e `Ctrl` + `w` `NomeArquivo`
salva como

`Ctrl` + `x` e `Ctrl` + `c`
Salva e sai do emacs.

Editar

`ESQ` + `@` `ou Mouse`
seleciona um bloco;

`Ctrl` + `w`
recorta o bloco selecionado;

`ESQ` + `w`
copia o bloco selecionado;

`Ctrl` + `y`
cola o bloco;

`Ctrl` + `d`
deleta o caracter sob o cursor;

`ESC` + `d`
deleta a palavra sob o cursor;

`Ctrl` + `k`
deleta a linha atual;

`Ctrl` + `w`
deleta os caracteres selecionados;

`Ctrl` + `x` + `u`
desfaz a última modificação;

Pesquisar

`Ctrl` + `s`
pesquisa por alguma string e avança para a próxima ocorrência;

`Ctrl` + `r`
pesquisa por alguma string e avança para a ocorrência anterior;

`ESQ` + `%`
Inicia o processo de substituição;

`Espaço`
Substitui e avança para a próxima ocorrência;

`Delete`
Não substitui e avança para a próxima ocorrência;

`!` Substitui todas as ocorrências;

`.` Substitui e sai do processo de substituição;

`ESQ` Sai do processo de substituição.

Navegação

`Alt` + `v` ou `PgUp`
Voltar uma página

`Ctrl` + `v` ou `PgDown`
Avançar uma página

`Ctrl` + `a` ou `Home`
Ir ao início da linha

`Ctrl` + `e` ou `End`
Ir ao fim da linha

`Ctrl` + `Home`
Ir início do documento

`Ctrl` + `End`
Ir ao fim do documento

`Ctrl` + `M` + `n`
Avança ao próximo parêntese que casa com aquele sob o cursor

`Ctrl` + `M` + `p`
Avança ao último parêntese que casa com aquele sob o cursor

Janela

`Ctrl` + `x` e `1`
Deixa apenas uma janela;

`Ctrl` + `x` e `2`
Divide a tela na horizontal;

`Ctrl` + `x` e `3`
Divide a tela na vertical;

`Ctrl` + `o`
Muda de janela;

D.3 Gedit

O Gedit é um editor de texto modo grafico, logo não existe muito segredo para editar, salvar ou abrir um arquivos. Mas é importante citar para aqueles, que preferem utilizar o modo gráfico.