PROJETO ESCOLAS - REFERÊNCIA
Compromisso com a Excelência na Escola Pública

Cadernos de Informatica

CURSO DE CAPACITAÇÃO EM INFORMÁTICA INSTRUMENTAL

CURSO DE MONTAGEM E MANUTENÇÃO DE COMPUTADORES

CURSO SOBRE O SISTEMA OPERACIONAL LINUX

CURSO DE PROGRAMAÇÃO EM JAVA

CURSO DE INTRODUÇÃO A BANCOS DE DADOS

CURSO DE CONSTRUÇÃO DE WEB SITES

CURSO DE EDITORAÇÃO ELETRÔNICA

CURSO DE ILUSTRAÇÃO DIGITAL

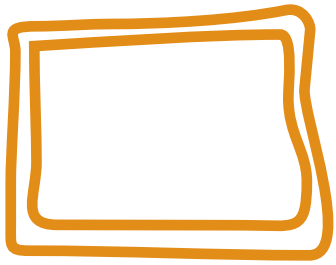
CURSO DE PRODUÇÃO FONOGRÁFICA

CURSO DE COMPUTAÇÃO GRÁFICA 3D

CURSO DE PROJETO AUXILIADO POR COMPUTADOR

CURSO DE MULTIMÍDIA NA EDUCAÇÃO

Cadernos de Informatica



CURSO SOBRE O SISTEMA OPERACIONAL LINUX

Edésio Costa e Silva

Coordenador:
Carlos Eduardo Hermeto de Sá Motta

Apresentação

Os computadores que estão sendo instalados pela SEE nas escolas estaduais deverão ser utilizados para propósitos administrativos e pedagógicos. Para isso, desenvolveu-se um conjunto de cursos destinados a potencializar a utilização desses equipamentos. São doze cursos que estão sendo disponibilizados para as escolas para enriquecimento do seu plano curricular. Esses cursos não são profissionalizantes. São cursos introdutórios, de formação inicial para o trabalho, cujo objetivo é ampliar o horizonte de conhecimentodos alunos para facilitar a futura escolha de uma profissão.

Todos os cursos foram elaborados para serem realizados em 40 módulos-aula, cada um deles podendo ser desenvolvidos em um semestre (com 2 módulos-aula semanais) ou em 10 semanas (com 4 módulos-aula semanais). Em 2006, esses cursos deverão ser oferecidos para os alunos que desejarem cursá-los, em caráter opcional e horário extra-turno.

Em 2007, eles cursos deverão ser incluídos na matriz curricular da escola, na série ou séries por ela definida, integrando a Parte Diversificada do currículo.

Esses cursos foram concebidos para dar aos professores, alunos e funcionários uma dimensão do modo como o computador influencia, hoje, o nosso modo de vida e os meios de produção. Para cada curso selecionado pela escola deverão ser indicados pelo menos dois ou, no máximo, três professores (efetivos, de preferência) para serem capacitados pela SEE. Esses professores irão atuar como multiplicadores, ministrando-os a outros servidores da escola e aos alunos.

Curso de Capacitação em Informática Instrumental

Este curso será implantado obrigatoriamente em todas as escolas estaduais em que for instalado laboratório de informática. Iniciando pelas Escolas-Referência, todos os professores e demais servidores serão capacitados para que possam fazer uso adequado e proveitoso desses equipamentos tanto na administração da escola como nas atividades didáticas.

É um curso voltado para a desmistificação da tecnologia que está sendo implantada. O uso do computador ainda é algo difícil para muitas pessoas que ainda não estão muito familiarizadas com essas novas tecnologias que estão ocupando um espaço cada vez maior na escola e na vida de todos. Este curso vai motivar os participantes para uma aproximação com essas tecnologias, favorecendo a transformação dos recursos de informática em instrumentos de produção e integração entre gestores, professores e demais servidores. As características dos equipamentos e as funcionalidades dos programas serão apresentadas de maneira gradual e num contexto prático. Essas.situações práticas serão apresentadas de maneira que o participante perceba o seu objetivo e o valor de incorporá-las ao seu trabalho cotidiano. Os participantes serão preparados

para navegar e pesquisar na internet; enviar, receber e administrar correspondência eletrônica, além de criar e editar documentos (textos, planilhas e apresentações) de interesse acadêmico e profissional. Esse é um curso fundamental, base e pré-requisito para todos os demais.

Curso de Montagem e Manutenção de Computadores

Este curso será implantado em, pelo menos, uma escola do município sede de cada Superintendência Regional de Ensino. A indicação da escola deverá ser feita pela própria S.R.E, levando-se em conta as condições de infra-estrutura nas Escolas-Referência existentes no município. Nas escolas escolhidas será montado um laboratório de informática especialmente para a oferta desse curso.

O objetivo deste curso é capacitar tecnicamente os alunos de ensino médio que queiram aprender a montar, fazer a manutenção e configurar microcomputadores. Pode ser oferecido para alunos de outras escolas, para professores e demais servidores da escola e para a comunidade, aos finais de semana ou horários em que o laboratório esteja disponível.

Neste curso o participante aprenderá a função de cada um dos componentes do microcomputador. Aprenderá como montar um computador e como configurá-lo, instalando o sistema operacional, particionando e formatando discos rígidos, instalando placas de fax/modem, rede, vídeo, som e outros dispositivos. Conhecerá, ainda, as técnicas de avaliação do funcionamento e configuração de microcomputadores que esteja precisando de manutenção preventiva ou corretiva, além de procedimentos para especificação de um computador para atender as necessidades requeridas por um cliente.

Dos cursos que se seguem, as Escolas-Referência deverão escolher pelo menos dois para implantar em 2006.

No período de 13 a 25 de março/2006, estará disponível no sítio da SEE (www.educacao.mg.gov.br) um formulário eletrônico para que cada diretor das Escolas-Referência possa informar quais os cursos escolhidos pela sua escola e quais os professores que deverão ser capacitados. Durante o período de capacitação, os professores serão substituídos por professores-designados para que as atividades didáticas da escola não sejam prejudicadas.

1. Curso sobre o Sistema Operacional Linux

É destinado àqueles que desejam conhecer ferramentas padrão do ambiente Unix. É um curso voltado para a exploração e organização de conteúdo. São ferramentas tipicamente usadas por usuários avançados do sistema operacional. Tem por finalidade apresentar alguns dos programas mais simples e comuns do ambiente; mostrar que, mesmo com um conjunto pequeno de programas, é possível resolver problemas reais; explicar

a comunicação entre programas via rede e estender o ambiente através de novos programas. O texto didático deste curso apresenta os recursos a serem estudados e propõe exercícios. É um curso para aqueles que gostam de enfrentar desafios.

Ementa: Histórico e desenvolvimento do Unix e Linux. Descrição dos conceitos de arquivo e diretório. Operações simples sobre arquivos e diretórios. Sistema de permissões e quotas.

Procurando arquivos. Executando e controlando programas. Processamnto de texto. Expressões regulares. Estendendo o ambiente. Trabalho em rede. Sistema de arquivos como um Banco de Dados.

2. Curso de Programação Em Java

É um curso de programação introdutório que utiliza a linguagem Java. Essa linguagem se torna, a cada dia, mais popular entre os programadores profissionais. O curso foi desenvolvido em forma de tutorial. O participante vai construir na prática um aplicativo completo (um jogo de batalha naval) que utiliza o sistema gráfico e que pode ser utilizado em qualquer sistema operacional. Os elementos de programação são apresentados em atividades práticas à medida em que se fazem necessários. Aqueles que desejam conhecer os métodos de produção de programas de computadores terão, nesse curso, uma boa visão do processo.

Ementa: Conceitos de linguagem de programação, edição, compilação, depuração e execução de programas. Conceitos fundamentais de linguagens de programação orientada a objetos.

Tipos primitivos da linguagem Java, comandos de atribuição e comandos de repetição. Conceito de herança e programação dirigida por eventos. Tratamento de eventos. Programação da interface gráfica. *Array*s. Números aleatórios.

3. Curso de Introdução ao Bancos de Dados

Este curso mostrará aos participantes os conceitos fundamentais do armazenamento, gerenciamento e pesquisa de dados em computadores. Um banco de dados é um repositório de informações que modelam entidades do mundo real. O Sistema Gerenciador do Banco de Dados permite introduzir, modificar, remover, selecionar e organizar as informações armazenadas. O curso mostra como os bancos de dados são criados e estruturados através de exemplos práticos. Ao final, apresenta os elementos da linguagem SQL (Structured Query Language – Linguagem Estruturada de Pesquisa) que é uma linguagem universal para gerenciamento de informações de bancos de dados e os elementos básicos da administração desses repositórios de informação..Apesar de ser de nível introdutório, o curso apresenta todos os tópicos de interesse relacionados à área. É um curso voltado para aqueles que desejam conhecer os sistemas que gerenciam volu-

mes grandes e variados de informações, largamente utilizados no mundo empresarial.

Ementa: Modelagem de dados. Normalização. Linguagem SQL. Mecanismos de consulta. Criação e alteração de tabelas. Manipulação e formatação de dados. Organização de resultados de pesquisa. Acesso ao servidor de bancos de dados. Contas de usuários. Segurança. Administração de bancos de dados. Manutenção. Integridade.

4. Curso de Construção de Web Sites

Este curso mostrará aos participantes como construir páginas HTML que forma a estrutura de um "site" na internet. A primeira parte do curso é voltada para a construção de páginas; a segunda parte, para a estruturação do conjunto de páginas que formação o "site", incluindo elementos de programação. Explicará os conceitos elementares da web e mostrará como é que se implementa o conjunto de páginas que forma o "site" num servidor.

Ementa: Linguagem HTML. Apresentação dos principais navegadors disponíveis no mercado.

Construção de uma página HTML simples respeitando os padrões W3C. Recursos de formatação de texto. Recursos de listas, multimídia e navegação. Tabelas e *Frame*s. Folha de Estilo. Elementos de Formulário. Linguagem Javascript. Interação do Javascript com os elementos HTML. Linguagem PHP. Conceitos de Transmissão de Site e critérios para avaliação de servidores.

1. Curso de Editoração Eletrônica

Voltado para a produção de documentos físicos (livros, jornais, revistas) e eletrônicos. Apresenta as ferramentas de produção de texto e as ferramentas de montagem de elementos gráficos numa página. O texto é tratado como elemento de composição gráfica, juntamente com a pintura digital, o desenho digital e outros elementos gráficos utilizados para promover a integração dos elementos gráficos.

O curso explora de maneira extensiva os conceitos relacionados à aparência do texto relativos aos tipos de impressão (fontes). Mostra diversos mecanismos de produção dos mais variados tipos de material impresso, de texto comum às fórmulas matemáticas. Finalmente, discute a metodologia de gerenciamento de documentos.

Ementa: Editor de textos. Formatadores de texto. Tipos e Fontes. Gerenciamento de projetos.

Publicações. Programas para editoração. Programas acessórios. Impressão. Desenvolvimento de um projeto.

2. Curso de Ilustração Digital

Desenvolvido sobre um único aplicativo de tratamento de imagens e pintura digital, o GIMP (GNU Image Manipulation Program – Programa de Manipulação de Imagens GNU).

Este curso ensina, passo a passo, como utilizar ferramentas do programa para produzir ilustrações de qualidade que podem ser utilizadas para qualquer finalidade. A pintura digital é diferente do desenho digital. O desenho se aplica a diagramas e gráficos, por exemplo. A pintura tem um escopo muito mais abrangente e é uma forma de criação mais livre, do ponto de vista formal. É basicamente a diferença que há entre o desenho artístico e o desenho técnico. É, portanto, um curso voltado para aqueles que têm interesses e vocações artísticas.

Ementa: A imagem digita*l.* Espaç*os* de core*s.* Digitalização de imagen*s.* Fotomontagem e colagem digital. Ferramentas de desenh*o.* Ferramentas de pintura. Finalização e saída.

3. Curso de Produção Fonográfica

Curso voltado para aqueles que têm interesse na produção musical. Explica, através de programas, como é que se capturam, modificam e agrupam os sons musicais para produzir arranjos musicais. É um curso introdutório com uma boa visão da totalidade dos procedimentos que levam à produção de um disco.

Ementa: O Fenômeno Sonoro. O Ambiente Sonoro. A Linguagem Musical. Pré-Produção. O Padrão MIDI. A Gravação. A Edição. Pós-processamento. Mixagem. Finalização.

4. Curso de Computação Gráfica

Curso introdutório de modelagem, renderização e animação de objetos tridimensionais.

Esse curso é a base para utilização de animações tridimensionais em filmes. Conduzido como um tutorial do programa BLENDER, apresenta a interface do programa e suas operações elementares. Destinado àqueles que têm ambições de produzir animações de alta qualidade para a educação ou para a mídia.

Ementa: Introdução à Computação Gráfica. Conceitos básicos 2D e 3D. Interface principal do programa Blender. Espaço de trabalho. Navegação em 3D. Modelagem em 3D. Primitivas básicas. Movimentação de objetos. Edição de objetos. Composição de cenas. Materiais e texturas. Aplicação de materiais. UV Mapping. Luzes e Câmeras. Iluminação de cena. Posicionamento e manipulação de câmera. Renderização still frame. Formatos de saída. Animação básica. Movimentação de câmera e objetos. Renderização da animação. Formatos de saída.

5. Curso de Projeto Auxiliado Por Computador

Os programas de CAD (Computer Aided Design – Projeto Auxiliado por Computador) são utilizados para composição de desenhos técnicos. Diferentemente dos programas de pintura eletrônica (como o GIMP), fornecem ao usuário ferramentas para desenhar com precisão e anotar os desenhos de acordo com as normas técnicas. Além de ensinar ao usuário a utilizar um programa de CAD (QCad), o curso apresenta elementos básicos de desenho técnico e construções geométricas diversas visando preparar o participante para um aprimoramento em áreas típicas das engenharias e da arquitetura..Ementa: Informática aplicada ao desenho técnico. Conceitos básicos: construções geométricas, escalas, dimensionamento, projeções ortográficas e perspectivas. Sistemas de coordenadas cartesiano e polar. Novas entidades geométricas básicas: polígonos e círculos.

Operações geométricas básicas. Tipos de unidades de medida. Criação de um padrão de

formato. Organização de um desenho por níveis. Construções geométricas diversas. A teoria dos conjuntos aplicada ao desenho. Propriedades dos objetos. Edição do desenho.

Movimento, rotação, escalamento e deformação de objetos. Agrupamento de objetos em blocos.

6. Curso de Multimídia na Educação

O curso está dividido em três partes: a) utilização da multimídia no contexto educacional; b) autoria de apresentações multimídia; c) projetos de aprendizagem mediada por tecnologia. Este curso é o fundamento para a criação dos cursos de educação a distância.

Apresenta os elementos que compõem os sistemas de multimídia, as comunidades virtuais de aprendizagem, o planejamento e a preparação de uma apresentação e de uma lição de curso e, finalmente, a tecnologia de objetos de aprendizado multimídia.

Ementa: Introdução à Multimídia e seus componentes. Multimídia na Educação. Comunidades Virtuais de Aprendizagem. “Webquest”: Desafios Investigativos baseados na Internet (Web).

Preparação de uma apresentação multimídia.

SUMÁRIO

Módulo 1

Preâmbulo

Bem-vindo ao curso de Introdução ao Sistema Operacional Linux!

O nosso objetivo é:

1. Antes de mais nada, interessar você em computação e desmistificar o acesso aos computadores
2. Definir o contexto dos sistemas Unix e Linux em particular
3. Apresentar alguns dos programas mais simples e comuns do ambiente
4. Mostrar que, mesmo com um conjunto pequeno de programas, é possível resolver problemas reais
5. Explicar a comunicação entre programas via rede
6. Estender o ambiente através de novos programas

Todo o material é apresentado em um ambiente apenas de texto. Se a interface que você dispõe é gráfica, utilizaremos um emulador de terminal.

Tentamos organizar o material como uma apresentação, seguida de exemplos, seguidos de exercícios. Os dois últimos módulos deste curso são quase desafios. Neles você poderá verificar a compreenção do material e resolver novos problemas.

Onde Podemos Chegar?

Com o material deste curso, mais curiosidade e um pouco de esforço, você será capaz de criar um programa capaz de gerenciar receitas. Veja como ele seria usado:

Procurando receitas usando um ingrediente:

```
mimbar:~:4> Receitas com ovos
001: Abóbora suíça
016: Misto quente com cobertura
018: Omelete Básica
020: Piperade
022: Pudim de abacaxi
025: Rolinhos de Siri
027: Salada Dinamarquesa
034: Torta de Fubá
mimbar:~:5> Receitas com chuchu
Ingrediente 'chuchu' nao encontrado em nenhuma receita!
```

Procurando receitas de um determinado autor:

```
mimbar:~:6> Receitas de Felipe
004: Cocada
005: Creme de banana e laranja
008: Frango empanado com queijo parmesão
010: Gelado de Morango
012: Lasanha aos Quatro Queijos
015: Manjar de coco com calda de amora
022: Pudim de abacaxi
023: Pudim de maria-mole enriquecido
024:
026: Salada alemã
028: Salada colorida
029: Salada da Copa
030: Salada de macarrão com brócolis
032: Sanduíche pizza
033: Gelado de Morango
034: Torta de Fubá
```

Listando uma receita específica:

```
mimbar:~:7> Receitas lista 024
nome: Quibe
Autor: Felipe Miranda
Autor: Cássia Rodrigues
Tipo: Lanche
Ingrediente: 500g de patinho moído
Ingrediente: 500g de trigo para quibe
Ingrediente: 01 cebola média picada
Ingrediente: salsinha e folhas de hortelã a gosto
Ingrediente: sal e pimenta-do-reino e malagueta a gosto
Preparo: Colocar o trigo de molho em água quente por 10 minutos.
Preparo: Espremer o trigo entre as mãos, retirando o excesso de água.
Preparo: Passar na máquina de moer carne  a cebola, a salsinha, o hortelã, o trigo e a carne por duas vezes (para a massa ficar bem fina).
Preparo: Temperar com o sal e as pimentas e enrole.
Preparo: Fritar em óleo quente.
```

Um programa parecido poderia ser criado para gerenciar livros, discos, CDs, fotografias, etc.

Interessa? Então, mãos a obra!

Introdução

Conceitos

O sistema operacional é o encarregado de gerenciar o hardware (equipamento) e disponibiliza-lo de maneira a ser utilizado pelo usuário e seus programas através de abstrações (o sistema operacional cria um ambiente simulado onde podemos enviar um documento à uma impressora sem precisar saber o tipo, modelo e forma de conexão da impressora). Ele intermedia os acessos ao hardware e pode prover recursos que não são diretamente disponíveis.

O que hoje chamamos de sistema operacional é composto de duas camadas:

- kernel – esta é a parte mais "básica" do sistema e, em princípio, a única que conversa com o hardware. Ele provê diversos serviços à próxima camada.
- utilitários – são os diversos programas fornecidos junto com o kernel através do qual o usuário interage com o kernel. No meio deles estão também programas de administração e de uso genérico e freqüente. Normalmente são pequenas aplicações.

Os utilitários podem ser divididos em duas grandes classes:

- os básicos, comuns a maioria dos sistemas e necessários administração do ambiente operacional. Normalmente são programas mais simples e com uma interface bem definida. Eles podem ser usados como "blocos de construção" para resolver problemas mais complexos.
- os de usuários, que são aplicações mais complexas como ambiente gráfico, editor de texto, navegação na internet (browser), jogos, etc.

O nosso enfoque será nos programas básicos.

Preliminares

Antes de alcançar o nosso objetivo, vamos às preliminares. Nas sessões abaixo temos uma referência rápida a alguns comandos e seqüencia de teclas que serão úteis durante o aprendizado. Quando você tiver uma dúvida, pode voltar aqui para procurar a resposta (não que todas as respostas estejam aqui...)

Comandos de teclado

Abaixo algumas sequencias de tecladas úteis. *Ctrl-n* quer dizer: pressione a tecla *Ctrl* e depois a tecla *n*. Solte a tecla *n* e a tecla *Ctrl*.

Ctrl-C cancelar um comando

Ctrl-D indicar fim de dados

Ctrl-U para apagar a linha corrente

Ctrl-S para suspender a impressão

Ctrl-Q para continuar a impressão

Caminhando na Lista de Comandos

Utilizando as setinhas do teclado é possível caminhar na lista de comandos recentemente digitados. Use

seta para cima para ver um comando anterior

seta para baixo para ver um comando posterior

seta para a esquerda move o cursor para a esquerda

seta para a direita move o cursor para a direita

Não são todos os programas que aceitam as setinhas. As ações acima valem para o `bash` e se seu terminal estiver corretamente configurado.

Comandos para Manipular Diretórios

pwd informa o diretório corrente

find procura um arquivo ou diretório no sistema de arquivos

cd muda de diretório

mkdir cria um diretório

rmdir remove um diretório vazio

Comandos para Manipular Arquivos

cp copia um (ou mais) arquivos

mv move (renomeia) um (ou mais) arquivos

rm remove um (ou mais) arquivos

Comandos para Manipular o Conteúdo de Arquivos

cat lista um (ou mais) arquivos

cmp compara dois arquivos

sort ordena as linhas de um (ou mais) arquivos

wc conta o número de linhas, caracteres e palavras de um arquivo

Comandos para Utilizar Disquete

mdir lista o conteúdo do disquete

mcopy copia arquivos de/para um disquete

mdel apaga arquivos em disquete

Notas

• Comandos longos para o shell podem ser divididos em múltiplas linhas terminando cada linha, exceto a última, com **** (barra invertida). Neste caso, o *prompt* muda para "> ".

• Múltiplos comandos podem ser colocados em uma mesma linha separados por **;** (ponto-e-vírgula).

• Apenas um pequeno conjunto de comandos e opções é descrito no texto. Maiores informações podem ser obtidas através do comando **man**. *Atenção:* a tradução das páginas dos manuais em português é, no mínimo, deficiente.

História

O Linux é uma versão de Unix. Existem outras, algumas "livres", disponíveis gratuitamente ou com baixo custo e com o código fonte liberado, por exemplo, o Minix, FreeBSD e OpenBSD. E existem as proprietárias, criadas ou mantidas por grandes fabricantes de equipamento ou software. Como exemplo temos o AIX da IBM, o Solaris da Sun e o HP-UX da HP. Foi inicialmente desenvolvido do "nada" por Linus Torvalds, em 1991. Naquela época, ele era um estudante na Finlândia. Ele colocou a versão 0.02 disponível na Internet e, a partir daí, muitas pessoas colaboraram com o seu desenvolvimento. Ainda hoje, Linus encabeça o projeto no papel de "ditador benevolente".

Já o Unix foi criado por Ken Thompson e Dennis Ritchie em 1969 no Bell Labs (laboratório de pesquisa da AT&T). Foi inicialmente criado para jogar Space Travel e rodava em um computador PDP-7 que estava abandonado num canto. O nome Unix veio de uma "brincadeira" com Multics. O Multics era/seria um sistema operacional muito pretencioso do qual a AT&T participou. O *mu* do nome significava Multiplexed (multiplexado, no caso, dividido em pedaços). Enquanto o Unix seria de um único pedaço.

Em 1973 o sistema foi reescrito para rodar em um computador de maior porte (para a época), o PDP-11/20. Agora, ao invés de linguagem Assembler, o sistema foi escrito em C. A idéia era que isto facilitaria o eventual transporte do sistema para novos computadores, e eles estavam certos. Desde o início o código fonte (programas) estavam disponíveis aos funcionários do Bell Labs. Uma conseqüência disto era que qualquer usuário que encontrasse um erro ou deficiência no sistema poderia saná-la.

Duas características marcaram o desenvolvimento do Unix desde o princípio:

- o pequeno porte da máquina disponível
- o fato dos primeiros usuários serem programadores.

O fato de a máquina ser pobre em recursos levou a criação de um sistema operacional "enxuto", contendo apenas o necessário para trabalhar de modo eficiente. Um exemplo disso é o fato das letras maiúsculas e minúsculas serem diferentes e significativas nos nomes dos arquivos (ao contrário do que acontece no Microsoft Windows). Simplesmente seria necessário mais código para compara-las caso fossem iguais.

Sendo os usuários programadores (e com todo o código do sistema sempre disponível) era fácil criar novos programas, corrigir os problemas e testar novas idéias. Bons programadores são também preguiçosos. Por isto, a maioria dos programas utilitários têm nomes compostos por duas ou três letras.

Distribuições

O "Ambiente" Linux é composto do kernel (sistema operacional) linux propriamente dito e um grande conjunto de programas. Dependendo de quem seleciona e "empacota" o ambiente são criadas "distribuições". Algumas das distribuições mais conhecidas são:

- Red Hat
- SuSE
- Debian

- Ubuntu
- Turbolinux
- Knoppix
- Conectiva/Mandrake/Mandriva
- Metasys

Cada uma delas tem uma peculiaridade ou foi criada para um tipo específico de usuário. Veja no glossário uma rápida descrição das distribuições.

Apesar das diversas distribuições existe um padrão para os programas disponíveis e sua localização no disco. Por isso, possível usar uma ou outra distribuição sem grandes transtornos. Os exemplos desta apostila foram executados em uma distribuição Debian. Os resultados podem ser diferentes no Metasys, que é a distribuição utilizada pela Secretaria de Educação. Em particular, as mensagens no Metasys devem aparecer em portugues.

Shell: Introdução

Quando se esta utilizando a console (o conjunto de monitor, teclado e mouse) ou um emulador de terminais (um programa, em um ambiente gráfico, que simula uma console), existe um programa que interpreta as requisições do usuário. Este programa é o *shell*. Shell em inglês quer dizer *casca* e faz sentido. O que você manipula numa ostra ou marisco é a concha. E ela protege o núcleo (kernel) do animal.

Um ponto interessante é que o **shell** não é um programa *especial*. É um programa de usuário como qualquer outro. Se você quiser pode escrever o seu próprio interpretador de comandos.

No Linux existem diversos programas shell. O que nós utilizaremos é o **bash** (**B**ourne **A**gain **SH**ell).

Existem duas formas de utilizar um shell:

- modo interativo — que veremos a seguir
- modo batch, script, lote ou não-interativo — que veremos em Shell: Avançado

Modo Interativo

O modo interativo é onde o usuário digita um comando e espera a execução dele antes de passar ao próximo comando. Um exemplo é quando se lista o conteúdo de um diretório. Você digita o comando e lê a listagem produzida.

Por exemplo, para saber que horas são podemos fazer o seguinte:

```
mimbar:~/local/wikit:2> date
Mon May 29 18:52:16 BRT 2006
mimbar:~/local/wikit:3>
```

date é o nome do programa que informa a data corrente.

A linha de comando

Pode parecer ultrapassado nos dias de hoje, com todos os recursos gráficos disponíveis, ainda utilizar a linha de comandos. Este é um ponto interessante. Existem, pelo menos, três bons motivos:

- a ferramenta (programa) gráfica pode não ser capaz de realizar a função que desejamos.
- podemos estar acessando o computador através de uma conexão de baixa velocidade (linha discada) e o ambiente gráfico não funciona bem nestas circunstancias.
- as aplicações gráficas podem ser muito **pesadas**, demandarem mais recursos do que o computador dispõe. O computador pode ser mais antigo, ter menos memória ou espaço em disco.

Vamos entrar em detalhes apenas sobre o primeiro motivo.

As ferramentas são ótimas quando utilizadas para resolver o problema para o qual foram desenvolvidas. Infelizmente elas falham quando o problema é um pouco diferente

do previsto. Um exemplo bem simples:

'*Você pode usar uma ferramenta gráfica, similar ao Windows Explorer, para descobrir todos os arquivos com um determinado nome ou extensão (.doc). Mas ela não é capaz de* contar *quantos arquivos foram encontrados. Você precisa fazer isto manualmente (se forem poucos arquivos). Outro exemplo, você quer renomear (mover) um conjunto de arquivos. A ferramenta pode fazer isto para você. Mas, e se você quiser colocar a data de hoje* antes *dos nomes de arquivos? Você vai ter que fazer isto arquivo a arquivo.*'

O grande "poder" do Unix, e por consequencia do Linux, é que existem diversas ferramentas (programas) que fazem uma função e a faz bem feita. E, através do shell, existe uma forma de combinar diversas ferramentas para resolver novos problemas.

Executando um programa

Na sua forma mais simples basta digitar o nome do programa que se quer executar. Veja o exemplo abaixo:

```
mimbar:~:7> date
Mon Aug 29 17:12:18 BRT 2005
mimbar:~:8>
```

Vejamos cada uma das linhas acima:

```
mimbar:~:7>
```

é o ***Prompt*** do shell. Ele indica que o shell esta pronto para receber um comando.

```
date
```

é o comando propriamente dito e

```
Mon Aug 29 17:12:18 BRT 2005
```

é o resultado da execução do comando.

```
mimbar:~:8>
```

O shell informa que esta pronto para um novo comando.

Caso o programa não exista, o shell envia uma mensagem de erro:

```
mimbar:~:27> data
-bash: data: command not found
```

Passando argumentos

Quando o shell recebe uma linha com um comando para executar, ele a divide em **palavras**. As palavras são seqüencias de caracteres separadas por seqüencias de espaços em branco. A primeira palavra é o nome do comando e as próximas (se existirem) os argumentos e opções.

Os argumentos modificam a execução dos comandos. Por exemplo, o comando **cal** (de calendario) precisa de dois argumentos, o mes e o ano a ser impresso:

```
mimbar:/usr/share/linux.see:20> cal 12 2005

    December 2005
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

Alguns dos argumentos podem ser *opções*. O primeiro caractere das opções é um **-** (hífen ou sinal de menos). As opções devem preceder os demais argumentos. Por exemplo, o comando **cal** aceita a opção **-m** faz com que o calendário seja listado com a segunda-feira na primeira coluna. Sem esta opção o domingo aparece na primeira coluna.

```
mimbar:/usr/share/linux.see:21> cal -m 12 2005
    December 2005
Mo Tu We Th Fr Sa Su
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Exercícios

Experimente digitar os seguintes programas, um por linha, terminando cada linha por ENTER:

- date
- whoami
- hostname
- echo “que lindo dia, nao?”
- pwd
- dir
- ls
- LS
- ls -a
- echo $SHELL

Os resultados esperados são:

- horário corrente
- nome do usuário
- nome da máquina
- o texto “que lindo dia, nao?” (sem as aspas). Observe que os espaços entre “que” e

"lindo" foram preservados! As aspas (") e os apóstrofes (') permitem utilizar textos com espaços como uma unidade (palavra). Veremos mais detalhes adiante.

- o diretório onde você está (provavelmente /home/usuario — veremos isto depois)
- talvez um erro, se o programa dir não existir
- a lista de arquivos do diretório corrente
- erro. As letras letras minúsculas e maiúsculas são diferentes no Unix. Os comandos são geralmente em minúsculas.
- a lista de arquivos do diretório corrente, incluindo os arquivos "ocultos"
- /bin/bash

Caracteres especiais

Você reparou que o comando *echo* "sumiu" com as aspas? E percebeu que o último comando não escreveu *$SHELL*? Existem alguns caracteres que são especiais para o shell. Eles são expandidos antes da execução da linha de comando. Os principais são:

$ (dollar ou cifrão) o identificador a seguir é o nome de uma variável

***** (asterisco) expande para uma lista de arquivos

~ (til) substituído pela variável HOME (o shell original **/bin/sh** não suporta o **~**)

? (interrogação) no nome dos arquivos, vale por qualquer caractere

` (acento grave) execução de comando

< (menor) redirecionamento da entrada padrão

> (maior) redirecionamento da saída padrão

| (barra vertical) composição de programas

**** (barra invertida) protege o próximo caractere

" (aspas) texto com substituição de variáveis (exemplo: "texto entre aspas")

' (apóstrofe) texto sem substituição de variáveis (exemplo: 'texto entre apóstrofes')

((abre-parentesis) sub-shell — termina com ")" (fecha-parentesis)

! (exclamação) acesso a história de comandos (apenas em modo interativo)

Veremos os detalhes mais adiante, na sessão de **Shell: Básico**. Por enquanto, se você precisar utilizar um dos caracteres especiais acima, coloque uma barra invertida (\) antes do caractere.

Exercício

- Repita os dois exercícios acima que não imprimiram o resultado esperado *protegendo* os caracteres especiais.

Sistema de Arquivos

O objetivo das tarefas propostas é explicar como utilizar o sistema de arquivos do Unix. Explicar o que é um arquivo, um diretório e como manipula-los.

Conceitos

O que é um arquivo?

O conceito de arquivo no Unix não difere muito do conceito de arquivo no mundo real. É uma coleção de informações. Estas informações podem ser homogêneas (terem sempre o mesmo formato) como as fichas de inscrição (todas iguais). Ou podem ser heterogêneas (cada qual de tamanho, cor, etc. diferentes) como um agregado de folhas com anotações.

Até o advento do Unix, os sistemas operacionais exigiam que as informações sobre os dados (que chamados de meta-data) fossem estabelecidas no instante da criação do arquivo e que fossem imutáveis. Em alguns sistemas as exigências eram ainda mais severas: apenas o administrador podia criar os arquivos e tinha que especificar onde ele seria armazenado, o tamanho de registro e o tamanho máximo do arquivo.

No Unix, o sistema se preocupa apenas em arrumar espaço em disco para o arquivo. Ele não faz nenhuma interpretação do seu conteúdo. Isto permitiu uma grande flexibilidade aos programas e usuários. Outra característica dos arquivos (e diretórios) no Unix é que eles possuem nome. Pode parecer estranho mas havia sistemas onde isso não acontecia.

Apesar de o Unix aceitar quase todos os caracteres no nome de arquivo, deve-se evitar o uso de caracteres acentuados, ponto-e-vírgula **(;)**, espaços em branco e tabulações, barra-invertida **(\)** e **&** (e comercial). Evite também arquivos começados por hífen ou menos **(-)**.

O caractere ponto (.) não é especial, no sentido de definir um tipo ou extensão para um arquivo. O nome *'sistema.de.arquivos.no.Unix'* é perfeitamenta aceitável. Nomes que começam por ponto (.), por convenção não são usualmente listados. Estes nomes são geralmente utilizados pelos arquivos de configuração dos programas. Mais exemplos em Introdução ao *Shell*.

Sistema de arquivos

Alguns sistemas operacionais organizam todos os arquivos do sistema em um balaio de gato. Todos os arquivos estão juntos. Isto funciona para sistemas pequenos onde são poucos os arquivos. Os disquetes são um exemplo.

A medida que o número de arquivos cresce fica cada vez mais difícil definir nomes únicos aos arquivos e encontrá-los. Além disso, dois usuários não podem ter arquivos com o mesmo nome.

Outros sistemas dividem os arquivos em duas partições: um conjunto de usuários e, para cada usuário, seus arquivos. Este tipo de organização inclusive diminui a interferência entre os diversos usuários. Esta estrutura era adotada pelo DOS/BATCH da Digital.

Os sistemas operacionais modernos possuem uma estrutura mais flexível onde, além de

arquivos em um diretório, podem existir também diretórios. Isto permite definir uma hierarquia para o armazenamento dos arquivos. Grosseiramente falando, você pode ter um diretório para a empresa, dentro dele um diretório para cada departamento, dentro de cada departamento, um diretório para cada sessão e assim por diante até chegar em diretórios que só possuem arquivos.

Os diretórios são chamados de pastas no Windows.

O que é um diretório?

Diretórios são utilizados para manter arquivos relacionados juntos. Podem ser arquivos executáveis (programas), arquivos de dados, arquivos de um projeto ou tema, arquivos de uma determinada data, etc. Distribuindo os arquivos em diretórios facilita a manipulação (cópia, acesso, compartilhamento) dos mesmos.

No Unix, e em alguns outros sistemas operacionais, os arquivos são organizados em diretórios que são, basicamente, arquivos com o nome de outros arquivos. No início do Unix, os mesmos programas que processavam arquivos texto podiam ser usados para processar diretórios. Depois, por questões administrativas e de segurança isto mudou. Agora os programas mkdir & rmdir são utilizados para criar e remover diretórios.

Estrutura de diretório

A estrutura de diretórios pode ser imaginada como uma árvore de cabeça para baixo.

O ponto de partida é chamado raiz (root) e representado por uma barra **(/)**. A partir daí estão os outros diretórios que, por sua vez, podem possuir outros diretórios.

Os diretórios abaixo de um diretório são chamados de filhos. O diretório imediatamente acima é chamado diretório pai. Um diretório pode ter inúmeros filhos e apenas um pai. O diretório pai é referenciado com o nome **..** (ponto-ponto).

No caminhar pela árvore de diretórios, usam-se os termos subir (ir para um diretório anterior (pai)) e descer (ir para um diretório contido no diretório em questão(filho)). Diretórios do mesmo nível e com o mesmo pai são chamados de irmãos.

Existe um padrão, nem sempre seguido, de estrutura de diretórios do Linux. Ela é descrita em Estrutura de diretórios porque é relativamente extensa.

Estrutura de diretórios

Em um sistema Unix típico existem centenas ou mesmo milhares de arquivos. Diversos destes arquivos são colocados em posições bem definidas de forma que as pessoas e os programas saibam onde achar diversas informações sobre o sistema.

A estrutura de diretório é organizada como uma árvore de cabeça para baixo, ou seja, com a raiz desenhada no nível mais alto e as olhas no nível mais baixo.

O caractere / (barra) é utilizado como separador dos componentes do nome do arquivo.

/ também é utilizada para representar o primeiro nível (raiz) da árvore.

Os principais diretórios são:

/ (raiz) início do sistema de arquivos

/bin utilitários mais comuns (essenciais)

/dev dispositivos do sistema

/etc informações e definições do sistema

/home diretório dos usuários

/lib bibliotecas mais comuns

/man manuais do sistema

/mnt ponto de montagem de sistemas de arquivos

/opt pacotes opcionais

/proc situação do sistema

/sbin utilitários para o administrador

/tmp diretório temporário

/usr aplicações/programas/utilitários de usuários

/var arquivos de trabalho

Nem todos estes diretórios precisam existir em todos os sistemas. Existe uma certa flexibilidade. E existe também a limitação do espaço em disco.

A seguir, descrevemos um pouco mais detalhadamente estes diretórios.

/ A imagem (kernel) do sistema operacional fica aqui. O diretório do administrador do sistema também.

/bin Os utilitários imprescindíveis, sem os quais o sistema não pode funcionar, como ls, cat, grep, cp,mv,rm, bash ficam neste diretório. Estes programas são usados pelo próprio sistema para manter a casa. Por exemplo, o comando rm é utilizado para remover os arquivos temporários do diretório /tmp.

/dev Os diversos dispositivos que o sistema suporta (impressoras, teclado, mouse, saída de som, entre outros) tem sua interface aqui. O arquivo /dev/lp0 é um nome comum para a primeira impressora do sistema.

Aqui estão também pseudo-dispositivos. Os mais famosos são /dev/null, utilizado para descartar a saída de um programa, e o /dev/zero que retorna um número infinito de nulos (zeros).

/etc As principais informações (estáticas) do sistema estão neste diretório. A maioria dos arquivos deste diretório são arquivos texto que podem ser editados pelo administrador do sistema.Por exemplo, a relação de usuários está no arquivo /etc/passwd.

Você pode lista-lo com o comando abaixo:

```
cat  /etc/passwd
  root:x:0:0:root:/root:/bin/bash
  sync:x:4:65534:sync:/bin:/bin/sync
  *** várias linhas suprimidas ***
```

Note que o nome e login do usuário estão neste arquivo mas não sua senha! A senha fica no arquivo /etc/shadow. Vamos lista-lo:

```
cat /etc/shadow

cat: /etc/shadow: Permission denied
```

Este arquivo é, por motivos óbvios, protegido contra a leitura e escrita. Um usuário comum não pode lê-lo.

Diversos arquivos de configuração dos programas do sistema estão aqui. O /etc/inetd.conf, por exemplo, contém os programas que devem ser disparados quando um determinado serviço (e-mail) é solicitado.

/home Este é o local onde tradicionalmente são criados os diretórios dos diversos usuários do sistema.

Muitas vezes este diretório está em um ou mais discos a parte permitindo assim que novos usuários possam ser adicionados ao sistema e também para aumentar o espaço em disco.

/lib Aqui ficam as bibliotecas, que são arquivos com várias funções dentro, que são utilizados por vários programas. As bibliotecas são parecidas com as bibliotecas do mundo real. Nelas são compartilhados os livros que são do interesse de diversas pessoas. Serve também como um repositório de conhecimento.

/man Os manuais dos programas, bibliotecas, arquivos de configuração, entre outros ficam neste diretório.

/mnt Atualmente caindo em desuso, o diretório /mnt, é o ponto de *montagem* de outros sistemas de arquivos. Por exemplo, para se utilizar um disquete no Unix ele pode ser montado em */mnt/floppy.*

/opt Grandes pacotes, opcionais, do sistema são comumente instalados aqui. Os sistemas de gerenciamento do ambiente gráfico (KDE e GNOME) às vezes são instalados aqui.

/proc Nem todo sistema Unix possui este diretório. Ele foi criado pela Sun para permitir o acesso, através do sistema de arquivo, à informações dinâmicas do sistema. Informações como o processador do sistema (/proc/cpuinfo), sistemas de arquivos suportados (/proc/filesystems) e dispositivos (/proc/devices) podem ser encontradas aqui. Ao contrário dos arquivos em /etc, as informações aqui mudam sem que o administrador precise edita-las.

/sbin Utilitários de uso restrito ao administrador, como criação de novos usuários, formatação de discos, ficam neste diretório ao qual normalmente os usuários não tem acesso.

/tmp Às vezes, durante a execução de um programa, é necessária a criação de arquivos intermediários (temporários) que não precisam serem preservados. Um bom local para criá-los é no /tmp. Periodicamente, normalmente durante a madrugada, ou quando a máquina é inicializada (boot) este diretório é apagado e recriado. Assim, todos os arquivos no /tmp são perdidos.

/usr Utilitários mais complexos (exemplo: latex) ou menos utilizados (nroff, agrep) e arquivos maiores (fonte do sistema) são colocados aqui. Muitas vezes o /usr é uma partição ou um disco a parte. Normalmente montado apenas para leitura. Dentro dele existem diversos subdiretórios:

/usr/bin mais utilitários

/usr/lib bibliotecas usadas pelos utilitários em /usr/bin

/usr/local arquivos instalados pelo administrador

/usr/share arquivos compartilhados (apenas leitura)

/usr/src fonte do sistema e seus utilitários

Quando diversos computadores estão ligados em rede, o administrador pode optar por compartilhar este diretório entre eles. Isto economiza espaço em disco e facilita a instalação de novos aplicativos. Basta fazer uma instalação e todos os computadores tem acesso aos novos aplicativos.

/var Nesta partição são colocados os arquivos e diretórios temporários que devem ser preservados quando a máquina é reiniciada. As mensagens de e-mail e os arquivos de log (registro de acontecimentos) ficam neste diretório.

Anteriormente os diretórios /usr e /var eram um só. Com o passar do tempo e o aumento do número de arquivos eles foram separados. Atualmente o /usr pode ser montado read-only (acesso apenas para leitura) e o /var montado para escrita. Esta separação dificulta a ação dos vírus, cavalos de Tróia e outros problemas de segurança e agiliza a inicialização (boot) da máquina.

Diretório corrente

Sempre que um programa é executado no Unix ele roda no diretório corrente, que é o local na hierarquia de diretórios onde o shell (e portanto o usuário) esta. Ele é referenciado por . (ponto). Quando um usuário entra (loga) no sistema ele usualmente vai para o seu diretório $HOME. A partir de lá, utilizando o comando do shell cd, ele pode se mover na estrutura de diretórios (árvore de arquivos).

```
mimbar:/usr/share/linux.see:7> pwd
/usr/share/linux.see
```

Recaptulando as convenções de diretórios:

. diretório corrente

.. diretório pai ou diretório acima

/ diretório raiz (origem)

~ diretório $HOME (nem todo shell ou programa aceita ~)

Referenciando arquivos e diretórios

Como foi dito acima, todos os arquivos e diretórios no Unix possuem, pelo menos, um nome. No texto que segue, não há diferenças entre arquivos e diretórios, portanto, para

simplificar, falaremos apenas de arquivos.

As referências aos arquivos podem ser de dois tipos:

• **absoluta** – quando o primeiro caractere do nome do arquivo é uma barra **(/)**, indicando que o nome deve ser analisado a partir do diretório raiz. Exemplo: /var/log/messages

• **relativa** – quando o nome do arquivo não começa com **/**. Neste caso, o arquivo deve ser procurado a partir do diretório corrente Exemplo: *.bash_profile*

Supondo que estamos no diretório */usr/share/linux.see* e que ele contém o arquivo citacao, todos os nomes abaixo acessam o mesmo arquivo:

```
mimbar:/usr/share/linux.see:9> ls - l citacao
-rw-r—r—  1 edesio edesio 42 Sep 13 18:48 citacao
mimbar:/usr/share/linux.see:10> ls - l ./citacao
-rw-r—r—  1 edesio edesio 42 Sep 13 18:48 ./citacao
mimbar:/usr/share/linux.see:11> ls - l ../linux.see/citacao
-rw-r—r—  1 edesio edesio 42 Sep 13 18:48 ../linux.see/citacao
mimbar:/usr/share/linux.see:12> ls - l /usr/share/linux.see/citacao
-rw-r—r—  1 edesio edesio 42 Sep 13 18:48 /usr/share/linux.see/
citacao
mimbar:/usr/share/linux.see:12> ls - l ./././././/citacao
-rw-r—r—  1 edesio edesio 42 Sep 13 18:48 ./././././/citacao
```

Outra caracteristica do sistema de arquivos do Unix é a homogeneidade. Todos os meios de armazenamento aparecem na mesma árvore. Não há o conceito de discos **(C:, D:)** do **DOS/Windows**. Disquetes, fitas, CD-ROMs, até uma impressora, aparecem como arquivos ou diretórios.

Permissões dos arquivos e diretórios

A partir do momento em que um computador é utilizado por mais de um usuário passa a ser importante definir quem pode fazer o que com os arquivos. E mesmo o que o dono do arquivo pode fazer com ele.

No Unix as permissões de acesso aos arquivos (e diretórios) são divididas em três conjuntos:

• o que o dono do arquivo pode fazer

• o que os usuários do mesmo grupo do dono pode fazer

• todos os outros usuários

São três as operações sobre os arquivos ou diretórios:

• **r** (Read — leitura)

• **w** (Write — escrita)

• **x** (eXecute — execução)

Para ilustrar veja a permissão do diretório **/** (raiz):

```
mimbar:/usr/share/linux.see:13> ls -ld /
drwxr-xr-x  21 root root 1024 Aug  5 15:59 /
```

- **'d'** para indicar diretório
- **'rwx'** são as permissões para o dono do diretório
- **'r-x'** as permissões para usuários do grupo do dono
- **'r-x'** as permissões para os outros usuários
- **'21'** é o número de sub-diretórios neste diretório
- **'root'** é o dono
- **'root'** é o grupo do dono
- **'1024'** é o número de bytes (caracteres) que o diretório ocupa
- **'Aug 5 15:59'** é a data da última alteração (escrita) neste diretório
- **'/'** o nome do diretório

Detalhes e exercícios sobre este importante assunto serão apresentados em *ls* e *chown & chmod*.

O administrador do sistema (root) não está limitado às permissões dos arquivos e diretórios!

Horários dos arquivos e diretórios

Além de permissões, os arquivos e diretórios no Unix possuem três horários (timestand):

atime – horário do último acesso ao arquivo (alterado quando se lê)

ctime – horário da última alteração de status do arquivo (alterado quando o arquivo é criado ou quando o dono ou as permissões são alteradas)

mtime – horário da última alteração do arquivo (escrita)

Estes horários podem ser exibidos através do programa *ls* e são úteis para fazer backups eficientes (copiando apenas os arquivos novos ou alterados). O comando *find* permite a identificação destes arquivos.

Observação: algumas implementações de sistemas de arquivos podem não implementar todos estes horários ou mante-los atualizados. Por exemplo, não é possível alterar o atime de um arquivo que está em um CD-ROM.

Note que o atime pode ser maior ou menor que o mtime. O shell utiliza isto para detectar a chegada de e-mail. Quando uma mensagem nova chega ela é escrita no arquivo da caixa postal do usuário e, portanto modifica o mtime. Se o *mtime > atime* existe uma mensagem nova e o shell informa isto ao usuário logo que possível. Quando o usuário lê a mensagem o atime é atualizado e agora temos *atime > mtime*.

Quotas de espaço em disco

Como o espaço em disco é compartilhado pelos usuários, pode ser importante restringir quanto de espaço em disco cada um pode utilizar. Para tal, são definidas quotas. Uma vez atingido o limite de sua quota, o usuário tem restrições sobre o espaço em disco que pode alocar.

Meio de armazenamento

O sistema de arquivo é muitas vezes referenciado como disco. Na quase totalidade das instalações, os arquivos são armazenados em discos rígidos, também chamados de winchesters e HDs. Os disquetes por sua vez são conhecidos como discos flexíveis). O meio físico onde esta o sistema de arquivo é transparente para o usuário, ou seja, ele não precisa saber o meio de armazenamento.

Sistema de Arquivos como Banco de Dados

O sistema de arquivos pode funcionar como um banco de dados (ver Glossário) em diversas situações. Para tal, o que precisamos é encontrar uma representação adequada aos dados. Voltaremos a este tópico mais adiante, após conhecermos os utilitários de manipulação de arquivos, diretórios e textos.

MÓDULO 2

SHELL: BÁSICO

Até agora vimos apenas como executar programas simples no Unix, programas que apenas imprimiam alguma coisa no terminal. Mas os programas podem também ler informações do terminal e emitir mensagens de erro.

Há também recursos no modo interativo para facilitar a vida do usuário. Veremos alguns destes recursos aqui.

ARQUIVOS PADRÃO

Quando um programa é executado no Unix ele "herda" três arquivos abertos:

- **stdin** – entrada padrão (descritor 0)
- **stdout** – saída padrão (descritor 1)
- **stderr** – saída de erros (descritor 2)

No modo interativo, o shell associa todos estes arquivos ao terminal do usuário. Mais adiante veremos como enviar o resultado do programa para um arquivo ou impressora e como ler os dados de um arquivo.

O shell é um programa como qualquer outro, lê a entrada padrão e escreve na saída padrão. Quando você digita o nome de um programa e o shell o executa, o shell para de ler e escrever e passa os arquivos padrão (entrada, saída e erros) para o programa. Tudo o que você digitar será lido pelo programa. E o que o programa escrever aparecerá na tela. Quando o programa terminar, o controle do computador retornará ao shell que voltará a ler e escrever no terminal. Vamos a um exemplo:

```
mimbar:~:2> wc
Procurando bem
Todo mumto tem pereba
Marca de bexiga ou vacina
E tem piriri, tem lombriga, tem ameba
^D
       2       18      101
mimbar:~:3>
```

O shell estava esperando um comando (indicado pelo *prompt* "mimbar:~:2> "), você digitou *wc*. O programa *wc* foi executado, leu as primeiras quatro linhas, o Ctrl-D (^D) que indica fim de texto, e imprimiu os númeors 4 (número de linhas no texto), 18 (número de palavras) e 101 (número de caracteres), e terminou. O shell voltou a esperar um comando (indicado pelo *prompt*).

Uma característica de muitos dos utilitários do Unix é que eles são escritos como *filtros*. Os Filtros lêem a entrada padrão, processam cada linha do arquivo e escrevem os resultados na saída padrão. Eventuais erros são enviados para a saída de erros.

Outros recursos que o shell oferece em modo interativo são: histórico de comandos e completar nomes.

Histórico de Comandos

O bash armazena os últimos comandos que você digitou. Através do recurso de histórico de comandos você pode reexecuta-los, com ou sem alteração. Isto é muito útil quando os comandos são mais longos.

Dependendo da configuração do seu terminal, as setinhas para cima e para baixo "andam" no histórico de comandos. As setinhas para a esquerda e para a direita "andam" dentro da linha. Para inserir um texto, basta digitá-lo. Para apagar um texto use a tecla DEL ou BACKSPACK ou ^U (Ctrl-U).

O comando history lista a relação de últimos comandos. Você pode repetir o último deles teclando **!!.** Pode repetir um comando arbitrário digitando !nnn onde nnn é o número do comando na lista. Ou pode utilizar !?palavra para repetir o último comando que contém a palavra palavra.

Veja abaixo um exemplo:

```
mimbar:~:12> cd /usr/share/linux.see
mimbar:/usr/share/linux.see:13> echo $PS1
\h:\w:\#>
mimbar:/usr/share/linux.see:14> date
Wed Dec 14 22:52:53 BRST 2005
mimbar:/usr/share/linux.see:15> ls
Bilhetes            assinatura.tcl   dominios.txt      rascunho.txt
Lista-de-Palavras.txt   campeoes.txt      doms          revision-1.txt
MaryPoppins-1.txt    capitais  frutas          revision-2.txt
MaryPoppins-2.txt    cidades   mlang10.zip     sigla-capital
Receitas             citacao   morse.txt       sigla-estado
assinatura.pl           dict-port.txt    mword10.zip
assinatura.sh           discurso.txt      pequeno.dicionario
mimbar:/usr/share/linux.see:16> history 5
 512   cd /usr/share/linux.see
 513   echo $PS1
 514   date
 515   ls
 516   history 5
mimbar:/usr/share/linux.see:17> !date
date
Wed Dec 14 22:53:14 BRST 2005
mimbar:/usr/share/linux.see:18> !?see
cd /usr/share/linux.see
mimbar:/usr/share/linux.see:19> !513
echo $PS1
\h:\w:\#>
```

history 5	lista os ultimos 5 comandos.
!date	reexecuta o ultimo comando date
!?see	reexecuta a ultima linha que contem o texto **see**

Cuidado com a repetição de comandos, principalmente o !? porque você pode estar no diretório errado para o comando!

Completar Nomes

Você pode digitar apenas o início do nome do programa ou arquivo e o shell o completa para você. Para isto você digita o início do nome e pressiona TAB. Se só existir uma alternativa o shell completa a palavra. Se existir mais de uma soará um beep. Neste caso, tecle TAB duas vezes para ver a lista de alternativas, digite mais alguns caracteres e pressione TAB novamente.

Quando se pressiona TAB na primeira palavra da linha de comando (o nome do programa a executar) o shell tenta fazer a expansão examinando cada alternativa na variável $PATH (esta e outras variáveis são descritas logo abaixo).

Após a primeira palavra o shell tenta expandir nomes de arquivos no diretório corrente.

Para cancelar a execução de um programa, pressione simultaneamente as teclas Ctrl e C.

Por exemplo, digite “cd /etc” (sem as aspas). Depois digite “echo pasTAB”. O shell completará o comando como “echo passwd”, porque o único arquivo que existe no diretório /etc que começa com “pas” é “passwd”. Experimente.

Observação: quando o shell expande um nome de diretório, com o recurso de completar nomes, ele coloca uma barra (/) para facilitar a digitação de um nome de arquivo.

Variáveis

Apertem os cintos que vamos ver um conceito um pouquinho mais complicado agora: variáveis.

Pode-se imaginar uma variável como uma caixa onde se pode guardar um texto. Como pode existir mais de uma caixa, cada caixa tem um nome único. Por exemplo, você pode ter uma caixa azul onde guarda a conta de telefone e uma caixa vermelha onde guarda o telefone da padaria.

Seguindo a metáfora da caixa, você pode pedir a alguém: “me passe a caixa vermelha” e “me passe o *conteúdo* da caixa vermelha”. São duas coisas bem diferentes.

No shell, para você colocar algo em uma variável você utiliza um comando da forma:

```
nome=value
```

Exemplo:

```
mimbar:~:117>  PADARIA="3261-3460"
```

Para obter o valor de uma variável você coloca um caractere dollar/cifrão (**$**) antes do nome da variável. Veja:

```
mimbar:~:118>  echo  PADARIA
PADARIA
mimbar:~:119>  echo  $PADARIA
3261-3460
```

Lá atrás, em Shell: Introdução, falamos dos caracteres aspas (“) e apóstrofes (‘). Veja como a expansão das variáveis se comporta:

```
mimbar:~:120> echo "$PADARIA"
3261-3460
mimbar:~:121> echo '$PADARIA'
$PADARIA
mimbar:~:122> echo "O telefone da padaria e' $PADARIA"
O telefone da padaria e' 3261-3460
```

e ainda,

```
mimbar:~:123> echo \$PADARIA
$PADARIA
```

Atenção: *não* pode haver espaços em branco ao redor do sinal de igual (=)!

E quando usamos variáveis? Quando queremos utilizar um texto mais de uma vez. Se vamos utilizar o mesmo texto diversas vezes, podemos coloca-lo em uma variável e referenciar a variável diversas vezes. Outra situação são nos comandos que veremos em Shell: Avançado.

Por fim, duas observações sobre variáveis:

- elas são efêmeras, ou seja, quando o shell termina (ou sua sessão) as variáveis são perdidas.
- a qualquer momento você pode alterar o valor de uma variável, afinal ela não é uma constante :-)

Variáveis do shell

O *shell* suporta e utiliza diversas variáveis. Algumas variáveis importantes são:

PS1 indicador que o shell esta pronto para receber um comando (*prompt*) — veja abaixo

HOME diretório principal do usuário

PATH relação de diretórios onde estão os programas

TERM definição do terminal

Além destas, existem outras variáveis definidas no ambiente que são utilizados por outros programas. Por exemplo,

EDITOR editor preferido pelo usuário (ver emacs e vi)

PAGER programa de paginação (ver more & less)

O usuário pode definir suas próprias variáveis, desde que não conflitem com as utilizadas pelo sistema. Por convenção, as variáveis do sistema são escritas sempre em maiúsculas.

As variáveis podem ser utilizadas para passar informações entre programas ou para armazenar valores temporários.

Você pode listar todas as variáveis definidas no ambiente através do programa 'env'. Ou listar individualmente com o comando 'echo $variável'. Exemplo:

```
mimbar:~:1# env
HZ=100
SHELL=/bin/bash
TERM=xterm
USER=root
MAIL=/var/mail/root
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin:/usr/bin/X11
PWD=/root
PS1=\h:\w:\##
SHLVL=1
HOME=/root
LOGNAME=root
_=/usr/bin/env
mimbar:~:2#
```

Ou

```
mimbar:~:2# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/bin/X11
mimbar:~:3#
```

O Prompt

Existem na realidade 4 prompts no shell:

PS1 – Este é o prompt primário. Existem algumas seqüencias de escape que são expandidas antes da sua exibição.

- **\h** – significa o nome do computador
- **\w** – significa o nome do diretório corrente
- **\#** – número do comando na história de comandos

PS2 – Utilizado para indicar que o comando está incompleto. O shell aguarda continuação.

PS3 – Exibido durante o comando [Select].

PS4 – Usado quando o shell está executando em modo de depuração.

Você pode examinar o conteúdo destas variáveis com o comando

```
mimbar:/usr/src:26> echo $PS1
\h:\w:\#>
mimbar:/usr/src:27>
```

E pode alterar com

```
mimbar:/usr/src:27> PS1="Pronto? "
Pronto? pwd
/usr/src
```

O valor inicial da variável PS1 (prompt) depende de cada distribuição. Pode ir desde um simples $ até prompts coloridos que ocupam mais de uma linha. Em todo caso, você pode alterar o valor desta variável na Configuração do Shell que veremos na sessão Shell: Avançado.

O *prompt* deve transmitir alguma informação ao usuário além de que o shell está pronto para o próximo comando. A definição do *prompt* nesta apostila é "\h:\w:\#> ". Isto permite saber em que máquina (**\h**) o usuário está (podem existir mais de um emulador de terminal aberto e cada um deles em uma máquina diferente), qual o diretório corrente (**\w**) e qual o número do comando corrente (**\#**). O número do comando corrente é útil para repetir um comando anterior pelo número (**!nnn**).

Entrada e Saída

Como dito acima, muitos programas recebem seus dados via entrada padrão e escrevem os resultados na saída padrão. Porém nem sempre isto é desejado, por exemplo, seus dados podem estar gravados em um arquivo. Para estas situações o shell implementa o *redirecionamento*. Através do redirecionamento o programa lê ou escreve em arquivos definidos pelo usuário. E isto sem que o programa perceba!

As formas mais comuns de redirecionamento são:

< da entrada padrão

> da saída padrão, sobrescreve

>> da saída padrão, adiciona

2> da saída de erros

| da saída padrão para a entrada padrão (pipe)

Vamos dar alguns exemplos utilizando os comandos **echo**, **cat**, **wc** e **date**. Estes comandos serão detalhados posteriormente. Rapidamente:

echo imprime os argumentos

cat lista o conteúdo de um arquivo

wc conta os linhas, palavras e caracteres de um arquivo

date imprime a data corrente

```
 mimbar:~/local/wikit/sh-exemplos:18> ls -l
 total 0
 mimbar:~/local/wikit/sh-exemplos:19> echo era uma vez...
 era uma vez...
 mimbar:~/local/wikit/sh-exemplos:20> echo era uma vez...
>historia
 mimbar:~/local/wikit/sh-exemplos:21> ls -l
 total 4
 -rw-r--r--  1 edesio edesio 15 Feb 19 21:36 historia
 mimbar:~/local/wikit/sh-exemplos:22> cat historia
 era uma vez...
 mimbar:~/local/wikit/sh-exemplos:23> wc historia
  1  3 15 historia
 mimbar:~/local/wikit/sh-exemplos:24> echo era uma vez... | wc
       1       3      15
 mimbar:~/local/wikit/sh-exemplos:25> cat <historia
 era uma vez...
 mimbar:~/local/wikit/sh-exemplos:26> date >>historia
 mimbar:~/local/wikit/sh-exemplos:27> cat historia
```

```
era uma vez...
Sun Feb 19 21:38:06 BRT 2006
mimbar:~/local/wikit/sh-exemplos:28> date >historia
mimbar:~/local/wikit/sh-exemplos:29> cat historia
Sun Feb 19 21:38:16 BRT 2006
mimbar:~/local/wikit/sh-exemplos:30> rm historia
mimbar:~/local/wikit/sh-exemplos:31> cat historia
cat: historia: No such file or directory
mimbar:~/local/wikit/sh-exemplos:32> cat historia 2>erros
mimbar:~/local/wikit/sh-exemplos:33> ls -l
total 4
-rw-r--r--  1 edesio edesio 41 Feb 19 21:44 erros
mimbar:~/local/wikit/sh-exemplos:34> cat erros
cat: historia: No such file or directory
mimbar:~/local/wikit/sh-exemplos:35> cd ..
mimbar:~/local/wikit:36> ls -l
CVS
Index.txt
Pages
dump.log
error.txt
history
linux.see
notes.tkd
rascunho.txt
savelog.txt
sh-exemplos
should-not-be-here-linux.see
tmp
wikit.copia
wikit.tkd
mimbar:~/local/wikit:37> ls -l | wc -l
15
mimbar:~/local/wikit:38>
```

O que aconteceu acima? Vejamos usando o número do comando como referência:

#18 o diretório está vazio

#19 usamos o *echo* que escreve na saída padrão

#20 redirecionamos a saída o *echo* para o arquivo historia

#21 dados do arquivo criado

#22 usando o *cat* para exibir o conteúdo do arquivo criado

#23 contando as linhas, palavras e caracteres do arquivo

#24 usando o **pipe** ao invés de um arquivo intermediário

#25 o *cat* listando a entrada padrão (porque foi executado sem argumentos)

#26 colocando a data atual no **final** do arquivo de historia (adicionado)

#27 novo conteúdo do arquivo historia

#28 sobrescrevendo o arquivo historia, o conteúdo anterior é perdido

#29 novo conteúdo do arquivo historia

#30 removendo o arquivo criado

#31 tentando exibir o conteúdo de um arquivo inexistente (erro)

#32 a mensagem de erro é escrita no arquivo *erro*

#33 conteúdo do diretório

#34 conteúdo do arquivo *erro*

#35 movendo para o diretório acima

#36 listando os arquivos do diretório

#37 contando os arquivos do diretório

Shell Scripts

Um *script* é um roteiro de comandos a executar. No caso de *shell script* o script é executado pelo shell. Existem outros interpretadores mas não entraremos em detalhes por agora.

Um shell script é um arquivo texto, marcado como executável (programa) e que tem na primeira linha **#!/bin/sh**. O shell lê e executa cada uma das linhas e exibe o resultado na tela.

Para que serve um shell script? Um shell script permite automatizar algumas funções. Vamos imaginar que você quer descobrir qual o arquivo mais recente de um diretório. Você pode utilizar os seguintes comandos:

```
mimbar:~:18> cd /tmp
mimbar:/tmp:19> ls -1t | head -1
plugtmp-10
mimbar:/tmp:20>
```

Ou pode criar um script com um editor de texto, como o vi, assim:

```
mimbar:/tmp:22> cat >mnovo
#!/bin/sh
cd $1
ls -1t | head -1
^D
mimbar:/tmp:23> chmod +x mnovo
mimbar:/tmp:24> ./mnovo /tmp
mnovo
mimbar:/tmp:25>
```

O script *mnovo* lista o arquivo mais recente do diretório passado como argumento.

Utilitários de manipulação de arquivos e diretórios

1. **ls**	listar conteúdo de diretório
2. **find**	procurar arquivos
3. **ln**	criar um "sinônimo" para um arquivo
4. **cp & mv & rm**	copiar, renomear e remover arquivos
5. **chown & chmod**	alterar dono e permissões dos arquivos
6. **file**	informar o tipo de um arquivo examinando seu conteúdo
7. **du & df**	utilização de espaço em disco
8. **mkdir & rmdir**	criar e remover diretórios
9. **cmp**	comparar dois arquivos

ls

ls é a abreviatura de list. Sua função é listar diretórios.

As principais opções do programa **ls** são:

-d lista as informações do diretório e não do seu conteúdo

-l lista além do nome, diversas informações sobre os arquivos

-t ordena a lista de arquivos pela data de última alteração

-S ordena a lista de arquivos por tamanho (decrescente)

-a lista os arquivos ocultos

Veja como fica a mesma listagem do diretório raiz (**/**) com as diversas opções:

```
 mimbar:/:2> ls
 bin   cdrom  etc   initrd      lib   media  opt   root  srv  tmp  var
boot  dev   home  initrd.img  lost+found  mnt   proc  sbin  sys  usr  vmlinuz
 mimbar:/:6> ls -a
 .     boot   etc      initrd.img  media  proc  srv  usr
 ..    cdrom  home     lib         mnt    root  sys  var
 bin   dev    initrd   lost+found  opt    sbin  tmp  vmlinuz
```

Por convenção, os arquivos que começam por ponto (**.**) não são usualmente listados. Muitos dos arquivos de configuração começam por (**.**). Veja em Introdução ao Shell.

```
 mimbar:/:3> ls -l
 total 60
 drwxr-xr-x 2  root root    2048 Aug 15 18:30 bin
 drwxr-xr-x 3  root root    1024 Aug 22 14:23 boot
 lrwxrwxrwx 1  root root      11 Aug  5 15:56 cdrom -> media/cdrom
 drwxr-xr-x 11 root root   24576 Aug 28 12:44 dev
 drwxr-xr-x 79 root root    4096 Aug 29 23:23 etc
 drwxrwsr-x 6  root staff    128 Aug 10 13:04 home
 drwxr-xr-x 2  root root    1024 Aug  5 15:57 initrd
 lrwxrwxrwx  1 root root      28 Aug  5 15:59 initrd.img -> boot/
initrd.img-2.4.27-2-386
 drwxr-xr-x 8  root root    4096 Aug 22 14:21 lib
 drwxr-xr-x 2  root root   12288 Aug  5 15:56 lost+found
 drwxr-xr-x 3  root root    1024 Aug  5 15:56 media
 drwxr-xr-x 15 root root    1024 Aug  5 21:32 mnt
 drwxr-xr-x 2  root root    1024 Aug  5 15:57 opt
 dr-xr-xr-x 121 root root 0 Aug 28 12:42 proc
 drwxr-xr-x 6  root root    1024 Aug 23 11:48 root
 drwxr-xr-x 2  root root    3072 Aug 22 14:21 sbin
 drwxr-xr-x 2  root root    1024 Aug  5 15:57 srv
 drwxr-xr-x 9  root root    0 Aug 28 12:42 sys
 drwxrwxrwt 9  root root    1024 Sep  2 17:48 tmp
 drwxr-xr-x 13 root root    1024 Aug  5 20:50 usr
 drwxr-xr-x 15 root root    1024 Aug  8 16:12 var
 lrwxrwxrwx  1 root root      25 Aug  5 15:59 vmlinuz -> boot/vmlinuz-
2.4.27-2-386
```

Na listagem longa os campos são os seguintes:

1. tipo do arquivo (l)
2. permissões do dono (rwx)
3. permissões do grupo (rwx)
4. permissões dos outros (rwx)
5. número de links
6. dono do arquivo
7. grupo do arquivo
8. tamanho em bytes do arquivo
9. data da última alteração
10. nome do arquivo
11. nome do destino (em caso de symlink)

O dono e as permissões podem ser alterados através dos comandos chown & chmod. Maiores detalhes no item Sistema de Arquivos.

```
mimbar:/:5> ls -Sl
total 60
drwxr-xr-x   11 root root  24576 Aug 28 12:44 dev
drwxr-xr-x    2 root root  12288 Aug  5 15:56 lost+found
drwxr-xr-x   79 root root   4096 Aug 29 23:23 etc
drwxr-xr-x    8 root root   4096 Aug 22 14:21 lib
drwxr-xr-x    2 root root   3072 Aug 22 14:21 sbin
drwxr-xr-x    2 root root   2048 Aug 15 18:30 bin
drwxr-xr-x    3 root root   1024 Aug 22 14:23 boot
drwxr-xr-x    2 root root   1024 Aug  5 15:57 initrd
drwxr-xr-x    3 root root   1024 Aug  5 15:56 media
drwxr-xr-x   15 root root   1024 Aug  5 21:32 mnt
drwxr-xr-x    2 root root   1024 Aug  5 15:57 opt
drwxr-xr-x    6 root root   1024 Aug 23 11:48 root
drwxr-xr-x    2 root root   1024 Aug  5 15:57 srv
drwxrwxrwt    9 root root   1024 Sep  2 17:48 tmp
drwxr-xr-x   13 root root   1024 Aug  5 20:50 usr
drwxr-xr-x   15 root root   1024 Aug  8 16:12 var
drwxrwsr-x    6 root staff   128 Aug 10 13:04 home
lrwxrwxrwx    1 root root     28 Aug  5 15:59 initrd.img -> boot/
initrd.img-2.4.27-2-386
lrwxrwxrwx    1 root root     25 Aug  5 15:59 vmlinuz -> boot/vmlinuz-2.4.27-2-386
lrwxrwxrwx    1 root root     11 Aug  5 15:56 cdrom -> media/cdrom
dr-xr-xr-x  120 root root      0 Aug 28 12:42 proc
drwxr-xr-x    9 root root      0 Aug 28 12:42 sys

mimbar:/:7> ls -lt
total 60
drwxrwxrwt    9 root root   1024 Sep  2 17:48 tmp
drwxr-xr-x   79 root root   4096 Aug 29 23:23 etc
drwxr-xr-x   11 root root  24576 Aug 28 12:44 dev
drwxr-xr-x    9 root root      0 Aug 28 12:42 sys
```

```
 dr-xr-xr-x  120 root root        0 Aug 28 12:42 proc
 drwxr-xr-x    6 root root     1024 Aug 23 11:48 root
 drwxr-xr-x    3 root root     1024 Aug 22 14:23 boot
 drwxr-xr-x    8 root root     4096 Aug 22 14:21 lib
 drwxr-xr-x    2 root root     3072 Aug 22 14:21 sbin
 drwxr-xr-x    2 root root     2048 Aug 15 18:30 bin
 drwxrwsr-x    6 root staff     128 Aug 10 13:04 home
 drwxr-xr-x   15 root root     1024 Aug  8 16:12 var
 drwxr-xr-x   15 root root     1024 Aug  5 21:32 mnt
 drwxr-xr-x   13 root root     1024 Aug  5 20:50 usr
 lrwxrwxrwx    1 root root       28 Aug  5 15:59 initrd.img -> boot/
initrd.img-2.4.27-2-386
 lrwxrwxrwx    1 root root       25 Aug  5 15:59 vmlinuz -> boot/
vmlinuz-2.4.27-2-386
 drwxr-xr-x    2 root root     1024 Aug  5 15:57 initrd
 drwxr-xr-x    2 root root     1024 Aug  5 15:57 opt
 drwxr-xr-x    2 root root     1024 Aug  5 15:57 srv
 lrwxrwxrwx    1 root root       11 Aug  5 15:56 cdrom -> media/cdrom
 drwxr-xr-x    3 root root     1024 Aug  5 15:56 media
 drwxr-xr-x    2 root root    12288 Aug  5 15:56 lost+found
 mimbar:/:4> ls -ld
 drwxr-xr-x   21 root root 1024 Aug  5 15:59 .
```

find

find quer dizer ache/procure. Este utilitário procura um ou mais arquivos ou diretórios dentro do sistema de arquivos. Ele é flexível para pesquisar por diversos atributos. A chamada tem esta forma:

```
mimbar:/usr/share/linux.see:23> find /usr -name "rascunho.txt"
/usr/share/linux.see/rascunho.txt
```

As principais opções são:

name padrão – procura por padrão

size tamanho – procura por arquivos de um determinado tamanho

xdev – não muda de sistema de arquivos

follow – segue os links simbólicos (symlinks)

type – tipo de arquivo (arquivo, diretório, symlinks, dispositivos)

maxdepth n – permite limitar a profundidade da pesquisa

Como já vimos, o espaço em disco é um recurso escasso. O administrador (root) pode querer saber todos os arquivos que ocupam mais de 10 MB (milhões de caracteres) nos diretórios dos usuários. Ele pode usar um comando como este:

```
mimbar:/usr/share/linux.see:3> find /home -size +10240k
/home/archive/tar/jre-1_5_0_04-linux-i586.bin
/home/archive/linux-2.6.12.3.tar.gz
```

Ou pode querer encontrar todas as cópias do firefox que estão em disco:

```
mimbar:/usr/share/linux.see:5> find / -type f -name firefox
/usr/lib/mozilla-firefox/firefox
```

No caso, só encontrou uma cópia.

Ou listar o primeiro nível de diretórios abaixo dos diretórios /usr e /var:

```
mimbar:/:108> find /usr /var -maxdepth 1 -type d
/usr
/usr/lost+found
/usr/share
/usr/bin
/usr/doc
/usr/games
/usr/include
/usr/lib
/usr/sbin
/usr/src
/usr/local
/usr/X11R6
/var
/var/lost+found
/var/lib
/var/cache
/var/backups
```

```
/var/local
/var/lock
/var/log
/var/run
/var/spool
/var/tmp
/var/opt
/var/mail
/var/www
mimbar:/:109>
```

updatedb/locate

Em sistemas de arquivos muito grandes, o tempo de execução do find pode ser considerável (vários minutos). Muitas instalações executam, durante a madrugada, um script chamado *updatedb* que percorre todo o sistema de arquivos e coleta o nome de cada arquivo (sem distinção de tipo). Durante o dia, os usuários podem utilizar o comando locate para encontrar os arquivos por padrão. As informações retornadas pelo locate podem estar desatualizadas, não contém os arquivos criados desde a execução do *updatedb* e ainda contém os arquivos que já foram removidos. Porém, a resposta dele é quase instantânea.

Usando o locate:

```
mimbar:/usr/share/linux.see:10> time locate '*/firefox'
/usr/bin/firefox
/usr/lib/mozilla-firefox/firefox
```

O locate reportou um arquivo a mais que o find (/usr/bin/firefox que é um symlink) porque ele não consegue distinguir os tipos de arquivos.
A título de exemplo, o comando find acima demorou mais de 9 minutos, enquanto o locate gastou 0.2 segundos.

Exercício

Quantos arquivos e diretórios existem na sua máquina? Utilize o comando wc (visto em Introdução ao Shell) para a contagem.

ln

No Unix em geral, e no Linux em particular, o conteúdo de um arquivo esta, num certo sentido, dissociado do seu nome. Como vimos no item Sistema de Arquivos, os diretórios armazenam o nome do arquivo e um apontador para o seu conteúdo. Então poderíamos ter dois (ou mais) nomes compartilhando o mesmo apontador e, portanto, o mesmo conteúdo. O ln realiza esta função.

ln vem de link, que quer dizer conectar. Na nossa área, pensamos mais em "apontar". O nome de um arquivo "aponta" para seu conteúdo.

Um arquivo pode ter um ou mais link. O programa ls exibe esta informação. Os diretórios também possuem links. Na realidade, possuem pelo menos dois:

- um vem do diretório "pai" (**..**) (que o antecede na hierarquia de diretórios)
- o outro vem dele mesmo (**.**) (cada diretório aponta para si mesmo)

Quando de "linka" um arquivo, o número de links é incrementado. Quando se remove um link (arquivo) [veja cp & mv & rm] o número é decrementado. Apenas quando o número de links chega a zero o conteúdo do arquivo é removido e a área em disco ocupada por ele liberada.

Só se podem criar links (hardlinks) para arquivos. Uma vez criado o link, o nome novo e o nome velho são indistinguíveis. Eles possuem todos os mesmos atributos (permissões, datas de acesso, dono, etc.).

Como o mesmo conteúdo é utilizado, o espaço em disco alocado para o arquivo não aumenta. Isto pode trazer ganhos significativos. Imagine que todos os usuários compartilham um arquivo (por exemplo, um dicionário de palavras-cruzadas). Basta que cada usuário tenha um link para ele.

Um dos usos de links é a criação de "sinônimos" para os arquivos. Por exemplo, um arquivo sobre a descrição de lentes pode ter os nomes otica e optica dependendo se a pessoa utiliza ou não o p mudo.

Outro uso é permitir acesso a um arquivo de acesso normalmente restrito. Por exemplo, o diretório $HOME pode ter acesso restrito mas o usuário quer publicar o arquivo tese.html, mas não os outros arquivos. Ele pode criar um link entre o arquivo tese.html e o arquivo tiotimolina.html na área de projetos do servidor web.

symlinks

Uma segunda forma de link existe: o symbolic link ou symlink. A grande diferença entre este tipo de link e o tipo descrito anteriormente (que seria o hardlink) é que, ao invés de apontar para o conteúdo do arquivo, ele aponta para o *nome* do arquivo.

Que impacto esta diferença traz?

1. o número de links não é mais alterado
2. arquivo pode ser apagado e recriado e o link continua funcionando
3. é possível criar um link para um diretório

No nosso exemplo do arquivo de palavras-cruzadas, dependendo de como o arquivo é atualizado, duas coisas podem acontecer:

1. o dono fica com o arquivo atualizado e todos os outros usuários (links) ficam com a versão anterior (hardlink) (existe uma forma de preservar os links, mas não entraremos em detalhes aqui)
2. todos vêem o mesmo arquivo (symlink)

cp & mv & rm

Inicialmente os três programas cp, mv e rm eram apenas um. O que acontecia era que haviam links com os três nomes. Depois esta situação foi alterada quando o comportamento deles foi formalizado.

Observe que as operações efetuadas por estes programas implicam na alteração (escrita) do diretório de origem e/ou destino! As permissões do diretório podem ser exibidas pelo comando ls e alteradas pelo programa chmod.

Os três programas aceitam uma opção comum: -i. Quando esta opção aparece, o programa questiona o usuário antes de executar uma ação irreversível (como apagar um arquivo). Para diminuir os acidentes é interessante sempre utilizar esta opção. Isto pode ser feito de diversas formas. Trataremos disto em Estendendo o Ambiente.

-r recursive (aplica a ação aos sub-diretórios)

-f force (executa a ação ignorando possíveis erros)

Veremos exemplos de uso mais abaixo.

cp

Este programa copia arquivos e diretórios.

Existem duas formas de execução:

- cp arquivo cópia
- cp arquivos diretório

Na primeira forma um único arquivo é copiado. Na segunda, quando é utilizada uma lista de arquivos, o último argumento deve ser um diretório.

Exemplos

- Existe um modelo do arquivo de configuração do bash no diretório /usr/share/linux.see com o nome de bashrc. Para copia-lo para o seu diretório $HOME use o comando:

```
cp /usr/share/linux.see/bashrc ~/.bashrc
```

Note que você copiou o arquivo com um outro nome (.bashrc).

- Você pode fazer cópias de arquivos importantes para poder voltar a traz em caso de problemas:

```
cp trabalho-de-quimica.txt quimica.20060201
```

No exemplo acima, o uso de uma data como parte (extensão) do nome do arquivo permite que os arquivos sejam listados em ordem cronológica.

- Quando copiar simultaneamente mais de um arquivo, o destino deve sempre ser um diretório. Observe:

```
mimbar:/usr/share/linux.see:27> ls -l cidades citacao
-rw-r--r--  1 edesio edesio 2851 Aug 30 14:06 cidades
-rw-r--r--  1 edesio edesio   42 Sep 13 18:48 citacao
mimbar:/usr/share/linux.see:28> cp cidades citacao local
cp: 'local': specified destination directory does not exist
Try 'cp --help' for more information.
mimbar:/usr/share/linux.see:29> cp cidades citacao /tmp
```

Exercícios

Você pode verificar se as ações foram executadas utilizando o programa ls

1. Copie o arquivo /etc/services para o seu diretório $HOME

2. Copie os arquivos /etc/services e /etc/hosts para $HOME/services

3. Copie os arquivos /etc/services e /etc/hosts para o diretório /tmp

O arquivo /etc/services contém a associação entre serviços de rede e número de porta/protocolo. Como será visto em Acesso a Rede. Já /etc/hosts contém a associação entre o nome de uma máquina e seu endereço IP (ver Acesso a Rede).

Quando se copia um arquivo, um novo arquivo é criado e portanto a hora de última alteração é o horário da execução do comando cp. Se for necessário preservar o horário de alteração do arquivo pode-se utilizar o comando tar.

mv

Move um arquivo de um diretório para outro. É assim que é implementado o renomear de um arquivo.

De forma similar ao cp o mv pode ser executado de duas formas:

- mv nome-velho nome-novo
- mv arquivos diretório

Exemplos

- Imagine que você fez uma grande bagunça com o arquivo do trabalho-de-quimica.txt e quer descartar as alterações feitas desde o dia primeiro de fevereiro:

```
mv -i quimica.20060201 trabalho-de-quimica.txt
```

Como você utilizou a opção -i o programa pede confirmação antes de sobrescrever o arquivo já existente.

- Os arquivos podem ser movidos de diretórios:

```
mv trabalho-de-quimica.txt /tmp
```

Se você executar este comando antes do comando anterior a versão bagunçada do arquivo será preservada no diretório /tmp.

Lembre-se que, conforme visto em Estrutura de diretórios, os arquivos do diretório /tmp são periodicamente apagados!

Internamente o mv é implementado como um ln seguido de um rm.

O mv não altera a data de última modificação de um arquivo porque o arquivo em sí (o conteúdo) não foi alterado. Os diretórios de origem e destino é que foram alterados!

Exercícios

1. Usando os arquivos copiados para o diretório /tmp no exercício acima, mude o nome do arquivo services para Servicos..

2. Tente fazer o mesmo com o arquivo no diretório /etc.

rm

Remove um arquivo. Como visto em ln, na realidade apenas decrementa o número de links do arquivo. Para se remover um arquivo não é necessário ter acesso de escrita ao arquivo. Mas é necessário ter acesso de escrita ao diretório que contém o arquivo.

Um exemplo de utilização do comando rm: o editor de texto emacs cria cópias backup dos arquivos editados. Estas cópias tem o mesmo nome do arquivo de origem seguido por ~. Se você editar o arquivo t.txt será criado um arquivo t.txt~ com o conteúdo do arquivo t.txt antes da edição. Depois de algum tempo editando arquivos o diretório terá vários arquivos da forma *~ Para remove-los podemos usar o comando

```
rm *~
```

*Cuidado! Se existir um espaço em branco entre o * e o ~ todos os arquivos do diretório corrente e do seu $HOME serão removidos!*

Quando um arquivo é removido, ele não pode ser recuperado. Portanto, pense bem antes de remover um arquivo. E mantenha backups dos arquivos importantes!

Para a remoção de diretórios deve ser usado o rmdir.

Uso das opções -f e -r

- Vamos fazer uma cópia de segurança (backup) do diretório de músicas, todos os sub-diretórios e os arquivos. Podemos usar o comando cp:

```
mimbar:/usr/share/linux.see:17> cp -r musicas/ /tmp/copia
```

- Dentro do diretório musicas, temos um arquivo de indice que, para evitar acidentes, está protegido contra escrita:

```
mimbar:/usr/share/linux.see/musicas:23> ls -l indice
-r—r—r—  1 edesio edesio 0 Jan 15 10:26 indice
```

Se tentamos simplesmente remove-lo temos uma mensagem de advertência:

```
mimbar:/usr/share/linux.see/musicas:24> rm indice
rm: remove write-protected regular empty file ‘indice’?
```

Se respondermos com y (yes/sim) o arquivo será removido. Porém, em um shell script, nem sempre teremos alguém para digitar a resposta. A alternativa é utilizar a opção -f.

```
mimbar:/usr/share/linux.see/musicas:25> rm -f indice
```

Outro detalhe da opção -f é que nenhuma mensagem de erro é emitida se o arquivo a ser removido não existir:

```
mimbar:/usr/share/linux.see/musicas:26> rm indice
rm: cannot remove ‘indice’: No such file or directory
mimbar:/usr/share/linux.see/musicas:27> rm -f indice
mimbar:/usr/share/linux.see/musicas:28>
```

chown & chmod

Estes dois programas (chown e chmod) modificam as informações sobre o arquivo (metadata) e não o conteúdo do arquivo.

chown

O chown modifica o dono do arquivo. Você pode criar um arquivo e passar a propriedade dele ao seu colega.

A sintaxe do comando é:

- **chown** dono arquivos
- **chown** dono diretórios

O comando aceita uma opção -R (recursivo). Quando esta opção é utilizada em um diretório, todos os arquivos e diretórios dentro dele são também alterados.

A função de alterar o dono de um arquivo é muitas vezes restrita ao administrador do sistema. Se não fosse assim, em um sistema de arquivos que implementasse quotas, um usuário poderia doar um arquivo para outro usuário, estourar a quota deste usuário e continuar utilizando o arquivo.

Um exemplo: o administrador do computador pode criar um usuário e um diretório *$HOME* para um visitante usar o sistema.

```
mimbar:/home:9# mkdir visita
mimbar:/home:10# ls -ld visita
drwxr-sr-x  2 root staff 48 Feb  1 21:07 visita
mimbar:/home:11# chown visita visita
mimbar:/home:12# ls -ld visita
drwxr-sr-x  2 visita staff 48 Feb  1 21:07 visita
```

Os comandos acima criam um diretório /home/visita e depois muda o dono para visita.

Na realidade existem comandos melhores e mais simples para realizar esta tarefa.

chmod

O chmod altera as permissões do arquivo. Apenas o dono ou o super-usuário podem alterar as permissões de um arquivo. Caso contrário, as restrições das permissões não seriam efetivas.

A sintaxe do comando é:

- chmod permissão arquivos
- chmod permissão diretórios

O comando aceita uma opção -R (recursivo). Quando esta opção é utilizada em um diretório, todos os arquivos e diretórios dentro dele são também alterados.

Através do programa ls podemos listar a permissão dos arquivos usando a opção -l:

```
mimbar:/usr/share/linux.see:66> ls - l dicionario
-rw-r--r--  1 root root 139462 Feb  1 21:00 dicionario
```

Vamos retirar a permissão de escrita do arquivo:

```
mimbar:/usr/share/linux.see:67> chmod a-w dicionario
chmod: changing permissions of 'dicionario': Operation not permitted
```

A mensagem de erro foi devido ao fato de não sermos o dono (owner) do arquivo. Apenas o dono do arquivo (ou o super-usuário) pode alterar a permissão de um arquivo.

Vamos fazer uma cópia do arquivo para o diretório temporário (/tmp) e repetir a operação:

```
mimbar:/usr/share/linux.see:72> cp dicionario /tmp
mimbar:/usr/share/linux.see:73> cd /tmp
mimbar:/tmp:74> ls -l dicionario
-rw-r--r--  1 edesio edesio 139462 Feb  1 21:03 dicionario
mimbar:/tmp:75> chmod a-w dicionario
mimbar:/tmp:76> ls -l dicionario
-r--r--r--  1 edesio edesio 139462 Feb  1 21:03 dicionario
```

Quando o arquivo foi copiado, ele foi criado pertencendo a nós. Assim podemos alterar as permissões dele.

Os arquivos com o bit 'x' ligado são chamados executáveis. O sistema só tenta executar arquivos com este bit ligado. Um "script" é um arquivo texto com o bit 'x' ligado. A primeira linha do arquivo define o programa que interpretará o "script".

MÓDULO 3

file

file é um programa que identifica o tipo de arquivo. No Unix, o nome de um arquivo não é associado a um tipo. Todos os arquivos são simples seqüencias de bytes.

No Windows, por exemplo, os arquivos .doc são documentos (Word ou de outro editor de textos). Os arquivos .bat são os equivalentes aos shell scripts do Unix. E os arquivos .exe são executáveis. No Unix, o arquivo avise-a-todos pode ser um programa executável, um shell script, um documento ou um diagrama elétrico.

file examina o conteúdo do arquivo a procura de características (assinaturas) que definam o seu tipo. Estas assinaturas são chamadas de magic numbers (números mágicos) e são um recurso de programação para evitar que um programa interprete um arquivo que não é para ele, o que poderia ter resultados desastrosos. Normalmente esses números mágicos estão no início do arquivo. Por exemplo, um arquivo com o bit de execução ligado (**x**) e que comece com **#!** é um script. O que vem após o **#!** é o nome do interpretador.

```
mimbar:/usr/share/linux.see:27> file *
assinatura.pl:          perl script text executable
assinatura.sh:          Bourne shell script text executable
assinatura.tcl:         a /usr/bin/tclsh script text executable
Bilhetes:               directory
campeoes.txt:           ISO-8859 text
capitais:               ASCII text
cidades:                ISO-8859 text
citacao:                ASCII text
discurso.txt:           ISO-8859 text, with very long lines
dominios.txt:           ASCII text
doms:                   ASCII text
Lista-de-Palavras.txt:  ASCII text
mlang10.zip:            Zip archive data, at least v2.0 to extract
morse.txt:              ISO-8859 text
mword10.zip:            Zip archive data, at least v2.0 to extract
pequeno.dicionario:     ASCII text
rascunho.txt:           empty
Receitas:               directory
sigla-capital:          ASCII text
sigla-estado:           ASCII text
```

Pode ser que esta lista de arquivos seja diferente da lista de arquivos da sua máquina!

du & df

Como foi visto no item Sistema de Arquivos, os usuários do sistema compartilham o espaço disponível para o armazenamento de arquivos.

Nos sistemas compartilhados, sejam eles Unix ou Windows, uma das mensagens mais freqüentes do administrador aos usuários é sobre a falta de espaço em disco.

Os administradores são, em geral, otimistas. O somatório das cotas de todos os usuários é maior que o espaço disponível. Isto funciona porque as chances de todos os usuários estarem utilizando toda a sua quota são pequenas. Para evitar situações de estouro da área em disco, o administrador pode lançar mão dos programas du & df.

df

df é a sigla para disk free e retorna o sumário da alocação de espaço em disco.

Na minha máquina de desenvolvimento, o resultado da execução do df é o seguinte:

```
mimbar:~/local/linux.see:13> df
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/hda2                466694    103781    338013  24% /
/dev/hda3                995932    534636    461296  54% /home
/dev/hda7                233335        18    220869   1% /tmp
/dev/hda5               1874267   1437345    336919  82% /usr
/dev/hda6                933361    418313    465249  48% /var
```

Como pode ser visto no cabeçalho do relatório, as colunas contém, na ordem:

- nome do disco ou partição
- tamanho da partição em milhares de caracteres (1K-blocks)
- espaço utilizado
- espaço disponível
- porcentagem do espaço utilizado
- mount-point (onde na hierarquia de diretórios está o sistema de arquivos)

Provavelmente a execução deste programa no seu computador exibirá resultados diferentes. Muitos administradores instalam todo espaço em disco em uma única partição (/). Isto simplifica um pouco a vida dele mas pode criar alguns problemas, que não detalharemos aqui.

Você pode passar um argumento para o "df" indicando de qual sistema de arquivos deseja a informação. Use "." para saber sobre o diretório corrente. Veja um exemplo:

```
mimbar:/usr/share/linux.see:13> df .
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/hda3                995932    918656     77276  93% /home
```

Observe que, apesar do diretório corrente estar em /usr o "df" listou as informações do /home. Isto quer dizer que existe um symlink em

```
/usr/share/linux.see (pode ser o /usr, o /usr/share ou o
/usr/share/linux.see).
```

Vamos criar um arquivo relativamente grande e ver como a quantidade de espaço em disco se comporta:

```
 mimbar:/usr/share/linux.see:14> dd if=/dev/zero of=arquivo-grande
bs=1k count=16000
 16000+0 records in
 16000+0 records out
 16384000 bytes transferred in 0.109046 seconds (150248387 bytes/sec)
 mimbar:/usr/share/linux.see:15> ls -l arquivo-grande
 -rw-r--r--   1 edesio edesio 16384000 May   1 13:01 arquivo-grande
 mimbar:/usr/share/linux.see:16> df .
 Filesystem            1K-blocks       Used Available Use% Mounted on
 /dev/hda3                995932     934672      61260  94% /home
 mimbar:/usr/share/linux.see:17> rm arquivo-grande
 mimbar:/usr/share/linux.see:18> df .
 Filesystem            1K-blocks       Used Available Use% Mounted on
 /dev/hda3                995932     918656      77276  93% /home
 mimbar:/usr/share/linux.see:19>
```

du

O du (disk usage — utilização de disco), num certo sentido, trabalha ao contrário do df. Ao invés de exibir o espaço livre exibe o espaço utilizado. Ele fornece também mais detalhes exibindo o espaço utilizado para cada diretório.

Este é um programa interessante para o usuário comum e não apenas para o administrador do sistema.

Novamente utilizando minha máquina como exemplo:

```
 mimbar:~/local/linux.see:21> du
0 ./Receitas
24      ./Bilhetes
6770    .
```

O número na primeira coluna é o número de KBytes em uso. A segunda coluna contém o nome do diretório. Observe que o espaço reportado para o diretório corrente (.) inclui o espaço ocupado pelos seus sub-diretórios (Receitas e Bilhetes). Isto fica mais visível em um diretório maior. Veja:

```
 mimbar:/usr/include:17> du
30      ./linux/netfilter_ipv6
27      ./linux/byteorder
3 ./linux/hdlc
75      ./linux/mtd
114     ./linux/netfilter_ipv4
19      ./linux/lockd
38      ./linux/dvb
95      ./linux/sunrpc
73      ./linux/nfsd
```

```
11     ./linux/netfilter_arp
43     ./linux/raid
30     ./linux/netfilter_bridge
30     ./linux/isdn
4437   ./linux
36     ./asm/mach-default
13     ./asm/mach-es7000
7 ./asm/mach-generic
22     ./asm/mach-visws
32     ./asm/mach-pc9800
15     ./asm/mach-summit
9 ./asm/mach-voyager
12     ./asm/mach-numaq
9 ./asm/mach-bigsmp
653    ./asm
82     ./asm-generic
52     ./arpa
```

Opções

Duas opções frequentemente utilizadas com o du são:

-s sumário — não lista os sub-diretórios individualmente

-h humano — imprime o espaço utilizado em unidades mais facilmente compreendidas pelos humanos

Exemplo:

```
mimbar:/usr/src:8> du -sh *
24K      Config-2.6.12
229M     linux-2.4.31
267M     linux-2.6.14.5
38M      linux-2.6.14.5.tar.bz2
274M     linux-2.6.15
48M      linux-2.6.15.tar.gz
31M      openswan-2.4.0
```

K representa kilobyte (1024 bytes = 1024 caracteres), M representa megabyte (1024 * 1024 = 1048576 bytes) e G representa gigabyte (1024 * 1024 * 1024 = 1073741824 bytes).

mkdir & rmdir

Como vimos em Sistema de Arquivos existem comandos próprios para a criação e remoção de diretórios o mkdir e rmdir.

mkdir

Novos diretórios podem ser criados para ajudar a organizar os arquivos do usuário. Por exemplo, pode-se criar um diretório para conter uma coleção de músicas organizadas por autor:

```
 mimbar:/usr/share/linux.see:12> mkdir musicas
 mimbar:/usr/share/linux.see:13> cd musicas/
 /usr/share/linux.see/musicas
 mimbar:/usr/share/linux.see/musicas:14> mkdir Wagner Bach Strauss
Beethoven
 mimbar:/usr/share/linux.see/musicas:15> ls -l
 total 0
 drwxr-xr-x   2 edesio edesio 48 Dec 15 09:09 Bach
 drwxr-xr-x   2 edesio edesio 48 Dec 15 09:09 Beethoven
 drwxr-xr-x   2 edesio edesio 48 Dec 15 09:09 Strauss
 drwxr-xr-x   2 edesio edesio 48 Dec 15 09:09 Wagner
```

Note que, a medida que você vai definindo novas categorias, você está criando uma árvore de diretórios. Você definiu uma hierarquia no conjunto de informações: primeiro os itens são classificados por tipo (música) e depois por autor. Se o número de autores fosse muito grande de forma que lista-los fosse inconveniente, poderia ser criado um nível intermediário, por exemplo, de estilo (clássico, jazz, funk, etc.).

Exercícios

- Armazene o conjunto das suas músicas favoritas em uma estrutura de diretórios organizado por estilo.
- Crie uma segunda organização destas mesmas músicas agora por compositor.
- Você consegue reutilizar os mesmos arquivos nas duas organizações?

Através de uma organização adequada dos arquivos e diretórios, é possível utilizar o sistema de arquivos como um banco de dados. Veremos isto mais adiante.

rmdir

O rmdir é a versão do rm capaz de remover diretórios. Em princípio o rm poderia ser programado de forma a tratar de forma homogênea arquivos e diretórios e, quando passado um diretório, remove-lo. Porém isto propiciaria acidentes. O usuário poderia remover um diretório quando desejava remover um arquivo.

Outro detalhe importante é que apenas diretórios vazios (que possuem apenas as entradas **.** e **..**) podem ser removidos.

Às vezes, a remoção de um diretório falha. Uma causa comum é a existência de um arquivo ou diretório oculto (começado por . (ponto)). Para verificar isto, execute:

```
mimbar:/usr/share/linux.see:36> ls -A
.backup  assinatura.pl      capitais doms mword10.zip   sigla-estado
Bilhetes assinatura.sh      cidades frutas         pequeno.dicionario
Lista-de-Palavras.txt  assinatura.tcl     citacao     mlang10.zip  rascunho.txt
MaryPoppins-1.txt  bashrc      dict-port.txt   modelo.txt revision-1.txt
MaryPoppins-2.txt bashrc~     discurso.txt   morse.txt  revision-2.txt
Receitas  campeoes.txt        dominios.txt   musicas    sigla-capital
```

Exercícios

- Crie uma estrutura de diretórios e remova aos poucos os arquivos e diretórios. Verifique a mensagem de erro quando se tenta remover um diretório não vazio.
- Você consegue remover um diretório para o qual você não tem permissão de escrita? Verifique sua resposta.

cmp

O cmp é um programa que compara dois arquivos e retorna se eles são iguais ou não.

Um dos usos deste utilitário é verificar se arquivos gerados por versões distintas de um programa são iguais ou se as alterações no programa causaram diferenças nos arquivos de saída. Ou para verificar se, por exemplo, o arquivo foi corretamente salvo (backup) em disquete ou CD. Veja o exemplo abaixo:

```
mimbar:/usr/share/linux.see:23> ls -l MaryPoppins-?.txt
-rw-r--r--  1 edesio edesio 36 Nov 27 12:59 MaryPoppins-1.txt
-rw-r--r--  1 edesio edesio 36 Nov 27 12:59 MaryPoppins-2.txt
mimbar:/usr/share/linux.see:24> cat MaryPoppins-1.txt
supercalifragilisticexpialidocious!
mimbar:/usr/share/linux.see:25> cat MaryPoppins-2.txt
supercalifragilisticexpialidocious!
```

Os dois arquivos parecem iguais. Você consegue indicar a diferença? Veja o que o cmp retorna:

```
mimbar:/usr/share/linux.see:27> cmp MaryPoppins-1.txt MaryPoppins-2.txt
MaryPoppins-1.txt MaryPoppins-2.txt differ: char 15, line 1
```

E também:

```
mimbar:/usr/share/linux.see:28> echo $?
1
```

Mais detalhes sobre o código de retorno em Shell: Avançado.

Exercício

• No diretório /usr/share/linux.see/Arquivos existem alguns arquivos com receitas simples. Você consegue verificar se alguma das receitas está repetida?

Apesar de existirem formas mais eficientes (md5sum), o cmp pode ser utilizado para verificar se arquivos importantes do sistema não foram alterados por vírus, outros programas ou outras pessoas.

Um programa similar, utilizado quando os arquivos são textos, é o diff.

Sistema de Arquivos: Exercícios

1. Imagine que existe um arquivo no sistema (/usr/share/novidades) com as últimas notícias da escola. Como você poderia fazer para exibir este arquivo mas somente se ele foi modificado desde a última vez que você o exibiu? Utilize o comando cat para exibir o arquivo.

2. O comando **date +%u** retorna o número correspondente ao dia da semana (1 para segunda-feira até 7 para domingo). Exiba o conteúdo do arquivo correspondente ao dia de hoje.

MÓDULO 4

PROCESSAMENTO DE TEXTOS

Aqui você utilizará os utilitários mais comuns para manipulação de arquivos texto.

O ambiente unix possui muitas ferramentas para manipular arquivos texto. Elas são tão versáteis que vale a pena fazer um pequeno esforço e utilizar arquivos texto ao invés de formatos proprietários. Em último caso, quando nenhuma ferramenta resolve de imediato o problema, você sempre pode entrar em um editor de textos (vi, emacs ou outro) e editá-lo.

1. **echo** imprime os argumentos
2. **cat** imprime o conteúdo de um arquivo
3. **head & tail** imprime as primeiras ou últimas linhas de um arquivo
4. **sort** ordena um arquivo
5. **cut** seleciona campos de um arquivo
6. **uniq** elimina linhas duplicadas em um arquivo
7. **tr** substitui caracteres
8. **wc** conta linhas, palavras e caracteres de um arquivo
9. **grep** procura por um padrão em um arquivo
10. **sed** edição programada de um arquivo
11. **awk** processamento de testos e padrões
12. **more & less** paginação de arquivo
13. **join** união de arquivos por um campo comum
14. **diff** comparação de dois arquivos

echo

Este programa, basicamente, imprime na saída padrão os seus argumentos. Exemplo:

```
mimbar:~:12> echo Quando       tudo foi dito e feito,     mais
foi dito que feito!
Quando tudo foi dito e feito, mais foi dito que feito!
mimbar:~:13>
```

Note que o número de espaços em branco entre as palavras é irrelevante e substituído por apenas um espaço.

Apesar de ser um dos programas mais simples do Unix, mesmo assim, ele possui opções. Duas delas, as mais usadas, são:

-n não imprime o fim-de-linha

-e interpreta seqüencias de escape (ex.: \t)

Veja a seguir:

```
mimbar:~:13> echo -n Hoje nasceu o filho de Deus!
Hoje nasceu o filho de Deus!mimbar:~:14>
```

Como não houve um fim-de-linha, o prompt ficou colado na saída do programa.

```
mimbar:~:15> echo -e A Terra \\te azul
A Terra          e azul
mimbar:~:16>
```

Entre 'Terra' e 'e' existe um caractere de tabulação (TAB) representado pelo caractere \t.

Vimos em Introdução ao Shell que o shell interpreta os argumentos antes de repassá-los ao programa. Por isso, foi necessário colocar duas barras invertidas (\\) porque a primeira delas foi consumida pelo shell.

Um uso comum do programa echo é na forma de diálogo com o usuário:

```
mimbar:~:16> echo "Qual o seu nome? "; read nome; echo "Voce respondeu
'$nome'"
Qual o seu nome?
Bond. James Bond
Voce respondeu 'Bond. James Bond'
mimbar:~:17>
```

Outro uso é em transformar uma lista de linhas em uma única linha. Veja:

```
mimbar:/usr/src:22> ls
Config-2.6.12   linux-2.4.31.tar.gz   linux-2.6.12.5
linux-2.4.31    linux-2.6.12.3        linux-2.6.12.5.tar.bz2
mimbar:/usr/src:23> echo `ls`
Config-2.6.12 linux-2.4.31 linux-2.4.31.tar.gz linux-2.6.12.3
linux-2.6.12.5 linux-2.6.12.5.tar.bz2
mimbar:/usr/src:24>
```

O comando ls lista os arquivos de um diretório. No caso, a saída deste comando ocupava 2 linhas. O comando echo a converteu em uma única linha.

cat

A função deste programa é a seguinte: dada uma lista de arquivos passada como parâmetros imprime o seu conteúdo na saída padrão.

Pode não parecer mas este utilitário é bastante útil. Ele permite, por exemplo, unir um arquivo com o cabeçalho, com o relatório, com o rodapé.

Exemplo (para evitar problemas de configuração, os arquivos de exemplo não possuem acentos):

```
mimbar:/usr/share/linux.see:6> cat capitais
RS=Rio Grande do Sul=Porto Alegre
SC=Santa Catarina=Florianopolis
PR=Parana=Curitiba
SP=Sao Paulo=Sao Paulo
MS=Mato Grosso do Sul=Campo Grande
MG=Minas Gerais=Belo Horizonte
GO=Goias=Goiania
MT=Mato Grosso=Cuiaba
TO=Tocantins=Palmas
ES=Espirito Santo=Vitoria
RJ=Rio de Janeiro=Rio de Janeiro
BA=Bahia=Salvador
RO=Rondonia=Porto Velho
SE=Sergipe=Aracaju
AL=Alagoas=Maceio
PE=Pernambuco=Recife
PB=Paraiba=Joao Pessoa
AC=Acre=Rio Branco
RN=Rio Grande do Norte=Natal
CE=Ceara=Fortaleza
PI=Piaui=Teresina
MA=Maranhao=Sao Luis
AM=Amazonas=Manaus
PA=Para=Belem
RR=Roraima=Boa Vista
AP=Amapa=Macapa
mimbar:/usr/share/linux.see:7>
```

Um dos argumentos do programa pode ser um ' - '. Neste caso, a entrada padrão será utilizada.

```
mimbar:~/local:32> echo Prezado senhor, ja sao >Header
mimbar:~/local:33> echo e o senhor continua logado no computador!
>Trailer
mimbar:~/local:34> date +%T | cat Header - Trailer
Prezado senhor, ja sao
18:06:55
e o senhor continua logado no computador!
mimbar:~/local:35>
```

Vamos examinar a execução do comando **date +%T | cat Header - Trailer** por partes:

1. Imprime o conteúdo do arquivo Header

2. Lê a entrada padrão, que neste caso é a saída do programa date, e a imprime

3. Imprime o conteúdo do arquivo Trailer

É mais simples do que parece!

Exercícios

- Você e seus colegas escreveram um trabalho de mecânica sobre máquinas primitivas. Cada um ficou com uma delas (cunha.txt, alavanca.txt, roldana.txt). Agora o professor quer que vocês entreguem um único arquivo com o trabalho. Como fariam isto?

- Se no exercício anterior for necessário acrescentar ao trabalho já pronto um prólogo e um epílogo como faria isto?

head & tail

Os comandos head e tail são, num certo sentido, complementares:

head lista as primeiras linhas de um arquivo

tail lista as últimas linhas de um arquivo

Ambos aceitam uma opção:

-n número de linhas a escrever

Quando nenhuma opção é fornecida, assume-se 10 linhas.

Exemplo:

- as primeiras 16 linhas do arquivo das 101 cidades mais populosas do mundo

```
mimbar:/usr/share/linux.see:14> head -16 cidades
2965000 | BUENOS AIRES | Argentina
4031000 | Sydney | Australia
3397000 | DHAKA | Bangladesh
10009000 | São Paulo | Brazil
2016000 | BRASILIA | Brazil
2139000 | Fortaleza | Brazil
2154000 | Belo Horizonte | Brazil
2331000 | Salvador | Brazil
5613000 | Rio de Janeiro | Brazil
2500000 | Toronto | Canada
4788000 | SANTIAGO | Chile
2051000 | Taiyuan | China
2101000 | Qingdao | China
2192000 | Changchun | China
2403000 | Jinan | China
2483000 | Dalian | China
```

- as últimas 8 linhas do mesmo arquivo

```
mimbar:/usr/share/linux.see:15> tail -8 cidades
1953000 | Houston (TX) | USA
2896000 | Chicago (IL) | USA
3694000 | Los Angeles (CA) | USA
8008000 | New York (NY) | USA
2589000 | KIEV | Ukraine
2142000 | TASHKENT | Uzbekistan
1823000 | CARACAS | Venezuela
3015000 | Ho Chi Minh City | Vietnam
```

Exercício

Imprima, a partir da linha de número 50, 5 linhas. O resultado deve ser:

```
7206000 | Delhi | India
9925000 | Bombay | India
1988000 | Medan | Indonesia
2080000 | Cirebon | Indonesia
2137000 | Sukabumi | Indonesia
```

Veja em *sort* um exemplo de head usando como filtro.

sort

Este utilitário coloca as linhas de um arquivo em ordem alfabética ou numérica.

Algum tempo atrás, estimava-se que 30% do tempo de CPU dos sistemas de grande porte era gasto na ordenação de arquivos. Toda vez que se deseja emitir um relatório ordenado por algum campo uma função de sort é utilizada.

Utilizando o arquivo de capitais como exemplo observe o resultado da execução do comando:

```
mimbar:/usr/share/linux.see:7> sort capitais
AC=Acre=Rio Branco
AL=Alagoas=Maceio
AM=Amazonas=Manaus
AP=Amapa=Macapa
BA=Bahia=Salvador
CE=Ceara=Fortaleza
ES=Espirito Santo=Vitoria
GO=Goias=Goiania
MA=Maranhao=Sao Luis
MG=Minas Gerais=Belo Horizonte
MS=Mato Grosso do Sul=Campo Grande
MT=Mato Grosso=Cuiaba
PA=Para=Belem
PB=Paraiba=Joao Pessoa
PE=Pernambuco=Recife
PI=Piaui=Teresina
PR=Parana=Curitiba
RJ=Rio de Janeiro=Rio de Janeiro
RN=Rio Grande do Norte=Natal
RO=Rondonia=Porto Velho
RR=Roraima=Boa Vista
RS=Rio Grande do Sul=Porto Alegre
SC=Santa Catarina=Florianopolis
SE=Sergipe=Aracaju
SP=Sao Paulo=Sao Paulo
TO=Tocantins=Palmas
mimbar:/usr/share/linux.see:8>
```

Quando nenhuma opção é fornecida, o programa ordena o arquivo em ordem alfabética crescente.

Algumas das opções disponíveis para o programa são:

-n ordena numericamente

-r coloca em ordem decrescente

-t terminador de campo (default é branco ou tab)

+n ordena pelo n-ésimo campo — os campos são numerados a partir de 0 (zero). O default é ordenar pelo registro inteiro, a partir do 0-ésimo campo.

Veja o que acontece quando se ordenam registros que começam por números (o arquivo possui as 100 mais populosas cidades do mundo). Os primeiros 10 registros do arquivo antes da ordenação, através do uso do comando head:

```
mimbar:/usr/share/linux.see:10> head cidades
2965000 | BUENOS AIRES | Argentina
4031000 | Sydney | Australia
3397000 | DHAKA | Bangladesh
10009000 | São Paulo | Brazil
2016000 | BRASILIA | Brazil
2139000 | Fortaleza | Brazil
2154000 | Belo Horizonte | Brazil
2331000 | Salvador | Brazil
5613000 | Rio de Janeiro | Brazil
2500000 | Toronto | Canada
```

Os primeiros 10 registros após a ordenação:

```
mimbar:/usr/share/linux.see:11> sort cidades | head
10009000 | São Paulo | Brazil
10231000 | SEOUL | South Korea
1823000 | CARACAS | Venezuela
1825000 | BUDAPEST | Hungary
1879000 | Kanpur | India
1885000 | Medellín | Colombia
1887000 | Mashhad | Iran
1929000 | Abidjan | Côte d'Ivorie
1953000 | Houston (TX) | USA
1988000 | Medan | Indonesia
```

Observe que o resultado está em ordem crescente comparado alfabeticamente (10009000 é menor que 10231000 porque 0 é menor que 2).

Agora utilizando a opção -n:

```
mimbar:/usr/share/linux.see:12> sort -n cidades | head
Population | City | Country
1823000 | CARACAS | Venezuela
1825000 | BUDAPEST | Hungary
1879000 | Kanpur | India
1885000 | Medellín | Colombia
1887000 | Mashhad | Iran
1929000 | Abidjan | Côte d'Ivorie
1953000 | Houston (TX) | USA
1988000 | Medan | Indonesia
1989000 | Quezon City | Philippines
```

A primeira linha Population | .. ficou em primeiro lugar porque, o primeiro campo não sendo numérico, foi comparado como 0 (zero).

Agora usando as opções -nr:

```
mimbar:/usr/share/linux.see:13> sort -nr cidades | head
10231000 | SEOUL | South Korea
10009000 | São Paulo | Brazil
9925000 | Bombay | India
9373000 | JAKARTA | Indonesia
9339000 | Karachi | Pakistan
8297000 | MOSKVA (Moscow) | Russia
8260000 | Istanbul | Turkey
```

```
8235000 | MEXICO (Mexico City) | Mexico
8214000 | Shanghai | China
8130000 | TOKYO | Japan
```

Exercício

Você trabalha numa empresa onde a central telefônica gera registros das chamadas realizadas. Estes registros são chamados bilhetes Por questões práticas, os bilhetes são gerados na ordem em que as ligações terminam. As primeiras linhas do arquivo /usr/share/linux.see/Bilhetes são:

```
mimbar:/usr/share/linux.see/Bilhetes:6> head -5 chamadas.txt
31/01/96 13:54 0201 01 2734155          000:05
31/01/96 13:55 0201 01 2612000          002:46
31/01/96 13:58 0201 01 2734155          001:19
31/01/96 14:01 0201 01 2741888          000:02 0242
31/01/96 14:01 0201 04 2748643          000:03 0243
```

O formato do arquivo é o seguinte:

1. Data de término da ligação
2. Horário de término da ligação
3. Ramal que efetuou a ligação
4. Tronco de saída utilizado
5. Número discado
6. Duração da chamada em minutos:segundos
7. Ramal para o qual a ligação foi transferida (se existir)

Como a conta telefônica tem um valor muito elevado, o seu chefe pediu que você organizasse as chamadas por ramais. Assim ele saberia quais ramais estão usando mais o telefone.

O resultado esperado é algo assim:

```
01/02/96 07:02 0201 01 90372612611      000:06
01/02/96 07:22 0201 02 2612579          000:08
01/02/96 07:23 0201 04 2611754          000:06 0205
01/02/96 07:32 0201 04 3541320          000:07
01/02/96 07:33 0201 05 3541320          000:21 0246
```

cut

cut é um filtro que extrai alguns campos das linhas de um arquivo texto.

O comando abaixo extrai o nome do país do arquivo de cidades mais populosas do mundo:

```
mimbar:/usr/share/linux.see:21> cut -d"|" -f3 cidades | head
 Argentina
 Australia
 Bangladesh
 Brazil
 Brazil
 Brazil
 Brazil
 Brazil
 Brazil
 Canada
```

Apenas as primeiras 10 linhas foram listadas.

As opções do cut são:

-d o caractere que delimita os campos

-f campos a serem extraídos

-c colunas a serem extraídas

O delimitador | (pipe) aparece entre aspas (") porque é um caractere especial para o shell.

Exercícios

• Liste a população e o nome da cidade.

O resultado deve ser:

```
2965000 | BUENOS AIRES
4031000 | Sydney
3397000 | DHAKA
10009000 | São Paulo
2016000 | BRASILIA
2139000 | Fortaleza
2154000 | Belo Horizonte
2331000 | Salvador
5613000 | Rio de Janeiro
2500000 | Toronto
```

• Você observou no exemplo acima que há um branco ' ' antes do nome do país? Como você pode retirá-lo?

• No diretório /usr/share/linux.see existem dois arquivos com uma relação de domínios (sites) da Internet. dominios.txt é o arquivo original que foi retirado do site http://registro.br/ Os nomes de domínios estão listados dois por linha. Como coloca-los um por linha?

```
mimbar:/usr/share/linux.see:26> head -5 dominios.txt
001design.com.br                001host.com.br
007fire.com.br                  007imoveis.com.br
00artoferotica.com.br           00hosting.com.br
01arquitetura.com.br            01dbstell.com.br
01dormitorio.com.br             01hospedagem.com.br
```

Um resultado pode ser:

```
001design.com.br
001host.com.br
007fire.com.br
007imoveis.com.br
00artoferotica.com.br
```

Dependendo do método utilizado, a ordem em que os dominios aparecem pode ser diferente.

Para conferir os resultados, verifique o número de linhas em cada arquivo:

```
mimbar:/usr/share/linux.see:33> wc -l doms dominios.txt
  59312 doms
  29656 dominios.txt
  88968 total
```

- Totalização de chamadas

O seu chefe o parabenizou pelo relatório que você gerou com as chamadas em sort. Porém, como todo chefe, ele não está satisfeito. O relatório é muito extenso e ele só quer saber quantas chamadas cada ramal fez. Você é capaz de resolver este problema?

Eu achei o seguinte:

```
   161 0201
     3 0202
    13 0203
    23 0205
    10 0207
```

- Ainda totalizando as chamadas

"Meu caro, gostei do relatório que você fez mas ele ainda me toma muito tempo. Eu quero só uma pequena alteração. Dá de listar em ordem decrescente de chamadas?"

Resultado esperado:

```
   161 0201
    23 0205
    20 0226
    13 0203
    11 0246
```

uniq

Este programa é utilizado para eliminar linhas adjacentes iguais, por exemplo, linhas em branco. Ou seja, deixa passar apenas as linhas únicas (unique).

No arquivo /usr/share/linux.see/campeoes.txt estão os times campeões mineiros de futebol entre 1950 e 2005.

```
mimbar:/usr/share/linux.see:42> head -5 campeoes.txt
Atlético Mineiro (Belo Horizonte)
Villa Nova (Nova Lima)
Atlético Mineiro (Belo Horizonte)
Atlético Mineiro (Belo Horizonte)
Atlético Mineiro (Belo Horizonte)
```

Utilizando-se o comando uniq podemos eliminar as linhas duplicadas.

```
mimbar:/usr/share/linux.see:43> head -5 campeoes.txt | uniq
Atlético Mineiro (Belo Horizonte)
Villa Nova (Nova Lima)
Atlético Mineiro (Belo Horizonte)
```

Com a opção -c podemos contar quantas vezes a mesma linha foi repetida:

```
mimbar:/usr/share/linux.see:44> head -5 campeoes.txt | uniq -c
      1 Atlético Mineiro (Belo Horizonte)
      1 Villa Nova (Nova Lima)
      3 Atlético Mineiro (Belo Horizonte)
```

Exercício

• Usando os comandos uniq, sort, head & tail, determine qual a mais longa seqüencia de vitórias de um só time.

Resultado:

```
      6 Atlético Mineiro (Belo Horizonte)
```

tr

tr vem de transliterate que é parecido, mas não igual, a tradução. Este utilitário troca um caractere por outro.

A sintaxe do comando é

tr conjunto#1 conjunto#2

Cada caractere do conjunto#1 é substituído pelo caractere equivalente do conjunto#2 Os conjuntos podem ser enumerados ou pode-se usar conjuntos pré-definidos:

abcdefghi	pode ser escrito como [a-i]
0123456789	pode ser escrito como [:digit:]
todas as letras	[:alpha:]
letras maiúsculas	[:upper:]
letras minúsculas	[:lower::]

Alguns caracteres especiais são especificados precedidos de barra invertida (\):

****	barra invertida
\n	new line (mudança de linha)
\r	carriage return (retorno de carro)

Observe que este utilitário não recebe nenhum nome de arquivo como argumento! Ele sempre processa a entrada padrão e envia os resultados para a saída padrão.

Exemplo, vamos transformar a letra **o** no número **0** (zero), a letra **i** no número **1** e **e** por **3.**

```
mimbar:/usr/share/linux.see:34> echo Belo Horizonte | tr oie 013
B3l0 H0r1z0nt3
```

Note que apenas as letras "oie" foram transformadas.

Exercício

Um dos mais antigos métodos de criptografia (mensagens secretas) é o "Alfabeto de César" que, diz a lenda, foi utilizado por Júlio César para enviar mensagens aos seus generais sem que elas fossem interceptadas pelo inimigo.

Funcionava substituindo uma letra pela terceira letra á sua direita. Exemplo:

```
abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz
defghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyzabc
```

Assim a palavra abacaxi era transformada em dedfdal.

Como você pode usar o tr para decifrar o texto abaixo, escrito com o "Alfabeto de César"?

```
pdlv vlpsohv gr txh sduhfh
```

Opções

O comando tr aceita as seguintes opções:

-c antes de começar, complementa o conjunto#1

-d remove todas as ocorrências do conjunto#1

-s substitui seqüencias do mesmo caractere uma só ocorrência

Podemos usar o tr com a opção -s para substituir seqüencias de brancos por _:

```
 mimbar:/usr/share/linux.see:36> echo "Quando a    luz dos
olhos meus encontra os     olhos teus" | tr -s ' ' _
 Quando_a_luz_dos_olhos_meus_encontra_os_olhos_teus
```

wc

wc significa word count, contador de palavras. É um utilitário simples que conta quantas linhas, palavras e caracteres possue um arquivo. Exemplo:

```
mimbar:/usr/share/linux.see:65> wc frutas
 73   80 744 frutas
```

As principais opções são:

-c retorna o número de caracteres

-l retorna o número de linhas

-w retorna o número de palavras

Exercício

• Utilizando os programas vistos até agora liste as palavras distintas do arquivo discurso. Considere que as palavras são terminadas pelos seguintes caracteres [,.?!] (vírgula, ponto, interrogação, exclamação e espaço em branco, os [] não fazem parte dos delimitadores). São esperadas 326 palavras.

• Continuando o exercício anterior, liste as palavras em ordem crescente de freqüência (use o comando sort). Algumas das palavras encontradas são:

```
      1 abater
      3 soldados
      4 hannah
      5 homem
     10 mundo
```

grep

O grep é um membro de uma família de utilitários que pesquisa palavras ou padrões dentro de arquivos. O nome grep vem de um comando do editor vi:

```
g/re/p
```

que é traduzido como:

```
global
regular expression
print
```

ou seja, pesquisa no arquivo todo, por uma Expressão Regular, e imprima as linhas onde ela foi encontrada.

Os outros membros da família são:

egrep extended grep que suporta um conjunto mais sofisticado de expressões regulares

fgrep fixed grep que pesquisa por cadeias de caracteres (strings) sem meta-caracteres

Existe um outro grep, agrep, que não faz parte dos utilitários distribuídos no Unix. Ele é de implementação muito mais recente. O a inicial quer dizer aproximado. Ao contrário dos outros o agrep tolera erros entre a expressão regular e o texto.

Opções

As principais opções dos programas grep, egrep e fgrep são:

-E interpreta a expressão regular como o egrep faria.

-F interpreta a expressão regular como uma lista de strings constantes, como o fgrep faria.

-G modo grep, apenas expressões regulares simples.

-h não imprime o nome do arquivo onde o padrão foi encontrado.

-i ignora a diferença entre MAIÚSCULAS e minúsculas.

-l lista apenas o nome do arquivo, sem exibir a linha onde o padrão foi encontrado.

-q como quieto. Não imprime nada, apenas informa via codigo de término (ver shell) se o padrão foi encontrado ou não.

-v inverte a seleção das linhas, ou seja, imprime as linhas que não possuem o padrão.

a versão do grep disponível no Linux é a do Projeto GNU. Esta versão se comporta como o egrep.

O programa fgrep possui algumas peculiaridades mas não teremos tempo de tratá-las.

Expressão regular

Uma expressão regular é uma forma compacta de definir algumas seqüencias de caracteres. Nós dizemos que uma expressão regular “casa” (ou “bate”) com um string quando o string satisfaz a descrição.

A primeira vista, as expressões regulares podem parecer ameaçadores mas, com um pouco de prática elas se tornam legíveis. Em todo caso, vamos começar pelas mais simples.

x onde x é um caractere sem significado especial, "casa" com o caractere. Este é o que chamamos de caractere literal.

. (ponto) "casa" com qualquer caractere

Para exemplificar vamos utilizar os comandos echo e grep:

```
mimbar:/usr/share/linux.see:15> echo 'linux' | grep 'i'
linux
```

Imprimiu 'linux' porque havia uma letra i na linha.

```
mimbar:/usr/share/linux.see:16> echo 'linux' | grep 'j'
mimbar:/usr/share/linux.see:17>
```

Não imprimiu nada porque não havia uma letra j na linha.

```
mimbar:/usr/share/linux.see:18> echo 'ruindows' | grep 'windows'
mimbar:/usr/share/linux.see:19> echo 'ruindows' | grep '.indows'
ruindows
mimbar:/usr/share/linux.see:20>
```

O primeiro comando não imprimiu nada porque não conseguiu achar 'windows' na linha. Já o seguinte procurou por qualquer caractere (.) seguido por 'indows'. Como este padrão foi encontrado, a linha foi impressa.

O próximo passo é definir uma seqüencia de alternativas para um caractere:

[aeiou] define que estamos procurando por qualquer um dos caracteres entre [e].

[^aeiou] define que estamos procurando por um caractere diferente dos que estão entre **[e]**. No caso, não apenas as consoantes mas também os caracteres de pontuação, espaço em branco, etc.

Se os caracteres estão na seqüencia alfabética você pode escrever [a-e] ao invés de [abcde].

Exemplo:

```
 mimbar:/usr/share/linux.see:20> echo 'Rei Luiz VIII' | grep
'Lui[sz]'
 Rei Luiz VIII
 mimbar:/usr/share/linux.see:21> echo 'Rei Luis VIII' | grep
'Lui[sz]'
 Rei Luis VIII
```

Ou seja, aceitamos Luis com 'z' ou 's'.

Exercícios

• Liste as linhas do arquivo /usr/share/linux.see/capitais onde duas vogais quaisquer aparecem juntas.

Você deve encontrar:

```
RS=Rio Grande do Sul=Porto Alegre
SC=Santa Catarina=Florianopolis
SP=Sao Paulo=Sao Paulo
MG=Minas Gerais=Belo Horizonte
GO=Goias=Goiania
MT=Mato Grosso=Cuiaba
ES=Espirito Santo=Vitoria
RJ=Rio de Janeiro=Rio de Janeiro
BA=Bahia=Salvador
RO=Rondonia=Porto Velho
AL=Alagoas=Maceio
PB=Paraiba=Joao Pessoa
AC=Acre=Rio Branco
RN=Rio Grande do Norte=Natal
CE=Ceara=Fortaleza
PI=Piaui=Teresina
MA=Maranhao=Sao Luis
AM=Amazonas=Manaus
RR=Roraima=Boa Vista
```

- No mesmo arquivo, liste os estados ou capitais que começam por vogal.

Você deve encontrar:

```
ES=Espirito Santo=Vitoria
SE=Sergipe=Aracaju
AL=Alagoas=Maceio
AC=Acre=Rio Branco
AM=Amazonas=Manaus
AP=Amapa=Macapa
```

- Agora liste apenas as linhas onde os estados começam por vogal.

Confira o seu resultado:

```
ES=Espirito Santo=Vitoria
AL=Alagoas=Maceio
AC=Acre=Rio Branco
AM=Amazonas=Manaus
AP=Amapa=Macapa
```

Ancoras

O próximo passo é introduzir "ancoras" na expressão regular. Elas definem onde a expressão deve bater. São elas:

^ início do string

$ fim do string

Vamos a um exemplo, listar as siglas de estado que começam por A:

```
mimbar:/usr/share/linux.see:13> grep ^A capitais
AL=Alagoas=Maceio
AC=Acre=Rio Branco
AM=Amazonas=Manaus
AP=Amapa=Macapa
```

Veja o que acontece se não ancorarmos o texto:

```
mimbar:/usr/share/linux.see:14> grep A capitais
RS=Rio Grande do Sul=Porto Alegre
BA=Bahia=Salvador
SE=Sergipe=Aracaju
AL=Alagoas=Maceio
AC=Acre=Rio Branco
MA=Maranhao=Sao Luis
AM=Amazonas=Manaus
PA=Para=Belem
AP=Amapa=Macapa
```

Exercícios

- Liste as capitais que terminam em consoante

Resultado esperado:

```
SC=Santa Catarina=Florianopolis
TO=Tocantins=Palmas
BA=Bahia=Salvador
RN=Rio Grande do Norte=Natal
MA=Maranhao=Sao Luis
AM=Amazonas=Manaus
PA=Para=Belem
```

Repetições e Opcionais

O que vimos até aqui é um subconjunto muito pequeno do que grep é capaz, todos os padrões de pesquisa tem comprimento ou tamanho constante. Existem operadores que eliminam esta restrição. São eles:

*	zero ou mais vezes
+	uma ou mais vezes (apenas no egrep)
?	opcional (apenas no egrep)

Vejamos como eles funcionam:

```
mimbar:/usr/share/linux.see:15> egrep 'a.*e' capitais
RS=Rio Grande do Sul=Porto Alegre
MS=Mato Grosso do Sul=Campo Grande
MG=Minas Gerais=Belo Horizonte
RJ=Rio de Janeiro=Rio de Janeiro
RO=Rondonia=Porto Velho
AL=Alagoas=Maceio
PE=Pernambuco=Recife
PB=Paraiba=Joao Pessoa
RN=Rio Grande do Norte=Natal
CE=Ceara=Fortaleza
PI=Piaui=Teresina
PA=Para=Belem
```

O padrão a.*e deve ser lido assim:

* **a** - deve conter uma letra a

* **.*** - qualquer caractere repetido zero ou mais vezes

* **e** - seguido de uma letra e

Examinando uma linha do resultado:

```
MG=Minas  Gerais=Belo  Horizonte
      a...e
            a....e
       a.........e
        a......................e
```

existem quatro posições onde o padrão pode ser encontrado. O grep é guloso (greedy) e procura o padrão mais a esquerda e mais longo, o último. Isto pode levar a resultados inesperados.

Exercícios

• No diretório /usr/share/linux.see existe o arquivo vocabulario.txt com uma relação de palavras em português. Como você poderia utiliza-lo para "trapacear" num jogo de forca?

• Escreva uma ou mais expressões regulares para encontrar alguns erros em textos em portugues. Por exemplo, depois de q sempre vem um u, usa-se m antes de b e p, alguns plurais terminam ns (e não ms).

• Em um caso mais complicado, verifique a existencia de letras duplicadas. Pares de r, s são válidos. Outros válidos como ee, oo são pouco frequentes e devem ser assinalados como erros.

sed

sed vem de stream editor, que quer dizer editor de "fluxo", "seqüencia". Não faz muito sentido em português. O que ele faz é aplicar alguns comandos de edição, originários do editor ed, em cada linha da entrada padrão e escrever o resultado na saída padrão.

Exemplo:

```
 mimbar:/usr/share/linux.see:50> echo 'Hoje é seu aniversário!' |
sed -e 's/Hoje é/Amanhã será/'
 Amanhã será seu aniversário!
```

No caso, o sed substituiu (comando s) **Hoje é** por **Amanhã será**.

O sed pode ser chamado como filtro ou passando a lista de arquivos como argumento.

A opção mais comum é:

-e script que executa o script para cada linha dos arquivos de entrada

O script por sua vez possui diversos comandos:

s/velho/novo/ substitute (substitui) a primeira ocorrência do texto velho (na realidade uma expressão regular) pelo novo.

s/velho/novo/g substitui todas as ocorrências de velho por novo

Um uso muito comum do sed é o de criar arquivos modificando um modelo (template):

```
 mimbar:/usr/share/linux.see:19> cat modelo.txt
 Olá!
 Nós da turma X, vamos nos reunir dia D, para comemorarmos o início das
férias.
 Cada participante deve levar um item. Você deve levar Q.
 Nos veremos lá.
 XXX
 mimbar:/usr/share/linux.see:24> sed -e 's/X/do funil/' \
       -e 's@D@7/9@' -e 's#Q#uma garrafa (cheia) de refrigerante#'
\
       -e 's/XXX/Maria Madalena/' modelo.txt
 Olá!
 Nós da turma do funil, vamos nos reunir dia 7/9, para comemorar-
mos o início das férias.
 Cada participante deve levar um item. Você deve levar uma garrafa
(cheia) de refrigerante.
 Nos veremos lá.
 do funilXX
```

Duas observações:

1. o caractere que vem após o s pode ser (quase) qualquer um. Ele não deve aparecer nem no texto velho nem no texto novo.

2. cuidado com a definição do padrão de pesquisa (texto velho). Veja na última linha o que aconteceu com a assinatura.

Exercícios

1. Verifique o que acontece no exemplo acima se você colocar a substituição da assina-

tura (XXX) antes das outras.

2. Mude o texto Maria Madalena para Charles Xavier e veja o que acontece.

3. Como você poderia garantir que a substituição sempre funcionasse?

Infelizmente a sintaxe das expressões regulares do sed é um pouco diferente das expressões regulares do grep ou mesmo do shell. Talvez a diferença mais significativa seja que os parentesis "(" e ")" devem ser precedidos pela barra invertida "\". Exemplo:

```
mimbar:~:10> echo "maracutaia" | sed -e 's/([aeiou])([aeiou])/XX/'
maracutaia
mimbar:~:11> echo "maracutaia" | sed -e 's\([aeiou]\)\([aeiou]\)/XX/'
maracutXXa
```

Outra diferença muito útil no sed é poder acessar o texto que "casou" com a expressão regular. Para isto, a expressão deve estar entre parentesis, "\(" e "\)", e deve ser referênciada como "\n", onde n é o número do grupo de parentesis. Parece confuso, vamos a um exemplo:

```
mimbar:~:12> echo "maracutaia" | \sed -e 's/\([aeiou]\)/>\1</'
m>a<racutaia
mimbar:~:13> echo "maracutaia" | \sed -e 's/\([aeiou]\)/>\1</g'
m>a<r>a<c>u<t>a<>i<>a<
```

Este recurso de referência a uma expressão regular anterior (back-reference) pode ser utilizado no texto velho. Vamos substituir "s" por "Z" quando aparecer entre duas vogais iguais:

```
mimbar:~:19> echo "casado" | sed -e 's/\([aeiou]\)s\1/\1Z\1/'
caZado
mimbar:~:20> echo "casulo" | sed -e 's/\([aeiou]\)s\1/\1Z\1/'
casulo
```

MÓDULO 6

awk

O awk é um programa e uma linguagem para processamento de textos criada em 1978 por Alfred Aho, Peter Weinberger, and Brian Kernighan.

Ao contrário dos outros programas que vimos até agora, o awk possui memória. Ao invés de processarmos um arquivo linha a linha, podemos agora lembrar o que aconteceu antes. Podemos imprimir apenas as linhas pares de um arquivo, por exemplo:

```
mimbar:/usr/share/linux.see:47> cat assinatura.sh
#!/bin/sh
# Primeiro colocamos cada letra em uma linha
# Depois ordenamos o arquivo
# Em seguida recolocamos tudo em uma linha so
# Entao tiramos os espacos em branco
# Finalmente imprimimos a palavra
echo ´
echo $1 | \
sed -e 's/./&\n/g' | \
sort´ | \
tr -d ' '
mimbar:/usr/share/linux.see:48> awk '(NR % 2)==0 {print}'
assinatura.sh
# Primeiro colocamos cada letra em uma linha
# Em seguida recolocamos tudo em uma linha so
# Finalmente imprimimos a palavra
echo $1 | \
sort' | \
mimbar:/usr/share/linux.see:49>
```

A chamada típica ao awk, como visto acima, é assim:

```
awk 'script' arquivo
```

onde script, entre apóstrofes para evitar conflitos de expansão de variáveis com o shell, é uma curta seqüencia de comandos.

Quando o awk processa um arquivo, ele decompõe cada linha em campos que são sequencias de caracteres separados por sequencias de espaços em branco (outros delimitadores podem ser utilizados mas não serão mencionados aqui).

Estes campos podem ser referenciados dentro do "script" como $1, $2 até $9. O campo $0 siginifica a linha inteira. Vamos listar a segunda palavra de cada linha do arquivo /usr/share/linux.see/citacao:

```
mimbar:/usr/share/linux.see:53> cat citacao
Um grande pais
se faz com grandes homens.
mimbar:/usr/share/linux.see:54> awk '{print $2}' citacao
grande
faz
```

Simples, não?

Exercício

• Imprima cada linha do arquivo (apenas os primeiros 9 campos) com os dois primeiros campos trocados. Exemplo: **um grande pais** seria impresso como **grande um pais**.

Você não conseguiria fazer isto com o comando cut porque ele imprime os campos na mesma ordem em que aparecem. Você pode apenas selecionar quais campos imprimir.

Outra variável que o awk define é NR que contém o número da linha corrente. Vamos numerar as linhas do arquivo citacao:

```
mimbar:/usr/share/linux.see:55> awk "{print NR ":", $0}' citacao
1: Um grande pais
2: se faz com grandes homens.
```

Espaços em branco (' ') são operadores de concatenação em awk.

Podemos, também, seguir cada linha com o seu comprimento:

```
 mimbar:/usr/share/linux.see:56> awk '{print $0 " (" length($0) "
caracteres)"}' citacao
 Um grande pais (14 caracteres)
 se faz com grandes homens. (26 caracteres)
```

A função length retorna o comprimento em caracteres de seu argumento. Exemplo: length("hoje") retorna 4.

Vamos a um exemplo que utiliza os recursos de memória do awk: imprima o comprimento da mais longa linha de um arquivo. Utilizando o arquivo cidades como entrada:

```
mimbar:/usr/share/linux.see:63> awk 'BEGIN {maior=0}
{if (length($0) > maior) { maior=length($0)}}
END { print "A maior linha tem " maior " caracteres"}' cidades
A maior linha tem 39 caracteres
```

A linha com BEGIN é executada antes da leitura do arquivo e a linha com END é executada depois da leitura do arquivo. A linha com if é executada para cada linha.

Exercícios

1. Modifique o script acima para imprimir também o número da linha mais longa.

2. Imprima apenas a linha mais longa.

3. Imprima a linha atual se ela for mais longa que todas as linhas anteriores. Ou seja, cada vez que você encontra uma linha mais longa que todas as que você viu antes você a imprime. Se a primeira linha for a mais longa de todas, você só imprime a primeira linha.

4. Como você imprimiria todas as linhas do arquivo em ordem crescente de comprimento? Sugestão: utilize os recursos de composição do shell e os comandos sort e cut.

more & less

Estes dois programas não são filtros. A função deles é paginar a exibição de um arquivo. Paginar significa que apenas um punhado de linhas é exibido de cada vez (normalmente o numero de linhas que o terminal comporta menos 2). A primeira linha é a última linha da tela anterior e a última linha é utilizada para interação com o usuário.

O primeiro programa escrito foi more, que em inglês quer dizer (envie) mais. Originalmente você apenas podia andar para frente no arquivo.

O segundo programa (less) é uma brincadeira com o more já que less significa menos. Ele é bem mais versátil que o more: permite que se volte em uma tela anterior, pesquisa por padrões (expressões regulares), disparar o editor de texto, etc.

join

join significa, em inglês, juntar ou unir. Este utilitário junta dois arquivos criando um terceiro. A peculiaridade deste programa é que ele une os dois arquivos de entrada através de um campo comum a eles. Talvez fique mais fácil com um exemplo.

No diretório /usr/share/linux.see existem dois arquivos, sigla-capital e sigla-estado como listados abaixo.

```
mimbar:/usr/share/linux.see:14> head -5 sigla-capital sigla-estado
==> sigla-capital <==
AC=Rio Branco
AL=Maceio
AM=Manaus
AP=Macapa
BA=Salvador
==> sigla-estado <==
AC=Acre
AL=Alagoas
AM=Amazonas
AP=Amapa
BA=Bahia
```

Observe que ambos estão ordenados pela sigla do estado. Através do utilitário join podemos unir estes dois arquivos e criar um novo onde, na mesma linha, aparecem a sigla do estado, seu nome e sua capital.

```
mimbar:/usr/share/linux.see:15> join -t = sigla-estado sigla-capital
| head
AC=Acre=Rio Branco
AL=Alagoas=Maceio
AM=Amazonas=Manaus
AP=Amapa=Macapa
BA=Bahia=Salvador
```

As principais opções deste programa são:

-t caractere delimitador de campo (usa branco se não for especificado)

-1 campo campo chave no primeiro arquivo

-2 campo campo chave no segundo arquivo

Os campos são numerados a partir de 1.

Este programa pode parecer quase inútil, mas a AT&T o usava como parte do sistema de registro e cobrança de chamadas telefônicas.

Exercício

Lá atrás, em sort e cut, o seu chefe pediu um relatório das chamadas realizadas. Hoje, talvez porque acordou com o pé esquerdo, ele veio cobrar mais uma pequena alteração:

Olha, o relatório que você gerou está muito confuso. Só tem um monte de números. Para analisa-lo vou precisar decorar toda a tabela de ramais. Que tal você imprimir o relatório com o nome dos ramais? Já pedi a secretária a relação de ramais e ela a colocou no arquivo /usr/share/linux.see/Bilhetes/ramais.txt. Vou tomar um cafezinho e volto para ver o novo relatório.

diff

Este comando exibe a diferença entre dois arquivos (ou duas versões do mesmo arquivo). É uma alternativa do cmp para arquivos texto.

No dia-a-dia parece que não há grande necessidade de um utilitário como este mas ele é muito importante quando se está escrevendo um programa ou um documento complexo. Ele se torna mais importante ainda quando mais de uma pessoa altera o mesmo arquivo.

Primeiro vamos a um exemplo da execução do diff:

```
 mimbar:/usr/share/linux.see:19> cat revision-1.txt
 Apesar do Unix aceitar quase todos os caracteres no nome
 de arquivo, deve-se evitar o uso de caractes acentuados,
 ponto-e-vírgula (;), espaços em branco e tabulações,
 barra-invertida (\). Evite também arquivos começados por
 hífen ou menos (-).
 mimbar:/usr/share/linux.see:20> cat revision-2.txt
 Apesar do Unix aceitar quase todos os caracteres no nome
 de arquivo, deve-se evitar o uso de caracteres
 acentuados, ponto-e-vírgula (;), espaços em branco e
 tabulações, barra-invertida (\) e & (e comercial). Evite
 também arquivos começados por hífen ou menos (-).
 mimbar:/usr/share/linux.see:21> diff -u -d revision-1.txt revision-
2.txt
 ——— revision-1.txt        2005-11-27 12:40:02.000000000 -0200
 +++ revision-2.txt        2005-11-27 12:40:13.000000000 -0200
 @@ -1,5 +1,5 @@
  Apesar do Unix aceitar quase todos os caracteres no nome
 -de arquivo, deve-se evitar o uso de caractes acentuados,
 -ponto-e-vírgula (;), espaços em branco e tabulações,
 -barra-invertida (\). Evite também arquivos começados por
 -hífen ou menos (-).
 +de arquivo, deve-se evitar o uso de caracteres
 +acentuados, ponto-e-vírgula (;), espaços em branco e
 +tabulações, barra-invertida (\) e & (e comercial). Evite
 +também arquivos começados por hífen ou menos (-).
```

A saída do programa, quando são usadas as opções -u e -d, deve ser interpretada da seguinte forma:

- as duas primeiras linhas, que começam com --- e +++ informam os arquivos comparados
- a próxima linha, que começa com @@ indica a faixa de linhas do primeiro e do segundo arquivos que aparecem abaixo
- as linhas seguintes contém o contexto e as alterações. As linhas que começam com um espaço em branco (‘ ‘) são o contexto das alterações, porque o mesmo conjunto de linhas pode aparecer diversas vezes. As linhas que começam com ‘-’ são linhas que foram removidas do primeiro arquivo. As que começam com ‘+’ foram adicionadas ao segundo arquivo.

Normalmente as diferenças não são tão pronunciadas, especialmente em programas. No exemplo acima, a alteração de caractes para caracteres causou alterações nas linhas seguintes. Isto não é típico.

Eu acho as opções -u e -d mais práticas. Sem as opções o resultado seria o seguinte:

```
mimbar:/usr/share/linux.see:22> diff revision-1.txt revision-2.txt
2,5c2,5
< de arquivo, deve-se evitar o uso de caractes acentuados,
< ponto-e-vírgula (;), espaços em branco e tabulações,
< barra-invertida (\). Evite também arquivos começados por
< hífen ou menos (-).
> de arquivo, deve-se evitar o uso de caracteres
> acentuados, ponto-e-vírgula (;), espaços em branco e
> tabulações, barra-invertida (\) e & (e comercial). Evite
> também arquivos começados por hífen ou menos (-).
```

- a linha 2,5c2,5 indica as linhas do primeiro e segundo arquivos do bloco de alterações abaixo.
- as linhas precedidas por < foram removidas no primeiro arquivo.
- a linha ------ separa as informações do primeiro e segundo arquivos.
- as linhas precedidas por > foram incluidas no segundo arquivo.

Observe que o programa reporta que encontrou alterações no seu codigo de retorno:

```
mimbar:/usr/share/linux.see:23> echo $?
1
```

Vimos em Introdução ao Shell que podemos testar este valor de retorno através do comando if.

MÓDULO 7

SHELL: AVANÇADO

Agora vamos ver recursos mais sofisticados do shell que permitem escrever verdadeiros programas em shell.

Gerando os Argumentos

Atente para o detalhe que a *entrada* do shell é texto e a *saída* dos programas (vários deles mas não todos) também é texto. Isto quer dizer que poderíamos utilizar a saída de um programa como entrada de outro, como vimos acima. Mas podemos também utilizar a saída de um programa como **argumento** de outro. Podemos fazer isto com o operador acento grave (`) que vimos na sessão de caracteres especiais.

O que este operador faz é passar o que um programa escreveu na saída padrão como argumentos de outro programa. Pode parecer confuso mas este é um conceito bem poderoso. Novamente vamos exemplificar utilizando os mesmos comandos da seção *Entrada e Saída*:

```
mimbar:~:15> cd /usr/share/linux.see/Bilhetes/
mimbar:/usr/share/linux.see/Bilhetes:16> ls -1
chamadas.txt
ramais.txt
mimbar:/usr/share/linux.see/Bilhetes:17> wc `ls -1`
  362  2276 17896 chamadas.txt
   20    58   402 ramais.txt
  382  2334 18298 total
mimbar:/usr/share/linux.see/Bilhetes:18> ls -1 >lista_de_arquivos
mimbar:/usr/share/linux.see/Bilhetes:19> cat lista_de_arquivos
chamadas.txt
ramais.txt
mimbar:/usr/share/linux.see/Bilhetes:20> wc `cat
lista_de_arquivos`
  362  2276 17896 chamadas.txt
   20    58   402 ramais.txt
  382  2334 18298 total
```

No diretório corrente temos dois arquivos, como pode ser visto pela execução do comando *ls -1* na linha #16. Na linha #17 chamamos o programa que conta linhas **wc** passando a lista de arquivos como parâmetro. Neste caso simples, poderíamos utilizar o comando abaixo:

```
mimbar:/usr/share/linux.see/Bilhetes:29> wc *
  362  2276 17896 chamadas.txt
   20    58   402 ramais.txt
  382  2334 18298 total
```

Em um exemplo mais complexo, vamos listar o número de linhas dos cinco arquivos mais recentes do diretório de Documentação da versão 2.6.15 do kernel linux (ufa!):

```
mimbar:/tmp:40> cd /usr/src/linux-2.6.15/Documentation
mimbar:/usr/src/linux-2.6.15/Documentation:41> ls -lt | head -5
total 1441
-rw-rw-rw-  1 root root  10581 Jan  3 01:21 00-INDEX
-rw-rw-rw-  1 root root   3699 Jan  3 01:21 BUG-HUNTING
-rw-rw-rw-  1 root root  13576 Jan  3 01:21 Changes
-rw-rw-rw-  1 root root  16241 Jan  3 01:21 CodingStyle
mimbar:/usr/src/linux-2.6.15/Documentation:42> ls -lt | head -5
00-INDEX
BUG-HUNTING
Changes
CodingStyle
DMA-API.txt
mimbar:/usr/src/linux-2.6.15/Documentation:43> wc `ls -lt | head
-5`
   302   1633 10581 00-INDEX
    92    539  3699 BUG-HUNTING
   438   1795 13576 Changes
   453   2636 16241 CodingStyle
   526   3028 20403 DMA-API.txt
  1811   9631 64500 total
```

Observe que, dentro dos acentos graves (`) temos mais de um comando. O resultado do *último* comando é passado como argumento.

Usamos o programa ***head*** *que será visto posteriormente. Ele lista as primeiras linhas de um arquivo.*

Estruturas de Controle

Normalmente os comandos em um script são executados na ordem em que aparecem. Para aumentar a flexibilidade dos scripts existem comandos no shell que alteram esta ordem. Os principais são:

if condicional – um comando é executado se uma determinada condição é satisfeita

for interação – um comando é executado para cada elemento de uma lista

while interação – um comando é executado enquanto uma determinada condição for válida

if

Vamos começar com um exemplo e depois discuti-lo:

```
echo -n 'Transfere R$ 1000 para a minha conta? (sim/nao)'
read resposta
if test "nao" = "$resposta"
then
     echo «Que pena. Eu estava contanto com este dinheiro»
else
     echo "Obrigado pela sua generosa doacao :-)"
fi
```

A estrutura deste comando é *if* condição *then* comando [*else* comando] *fi*. A parte entre '[' e ']' é opcional. Se a condição for verdadeira o comando (pode ser mais de um) que

segue o *then* é executado. Se a condição for falsa e a parte do *else* existir, o comando associado é executado.

condição é o resultado da execução de um programa. Diversos programas podem ser utilizados sendo que um dos mais comuns é **test**.

Observe que o teste é se o valor da variável resposta é *não*. **Qualquer** outro valor para a variável resposta será interpretada como *sim*!

for

Novamente um exemplo:

```
for sobrinho in Huginho Zezinho Luizinho
do
     echo "$sobrinho diz: Boa noite, Tio Donald!"
done
```

A sintaxe do comando é *for* (para) variável *in* (em) lista de valores *do* (faça) comando *done*. *comando* é executado com a variável tomando cada valor da lista. O texto entre *(* e *)* é uma tradução do termo em inglês.

A lista de valores é definida **antes** da execução do comando *for* e permanece inalterada. O final da lista de valores é definido pelo fim da linha de comando ou um ponto-e-vírgula (";").

while

A sintaxe do comando é *while* **condição** *do* comandos *done*.

while significa *enquanto* e os comandos entre o *do* e o *done* são executados enquanto a condição for verdadeira. Um exemplo bem simples:

```
 mimbar:/usr/share/linux.see:80> contador=0
 mimbar:/usr/share/linux.see:81> while test $contador -lt 5; do
echo "contador=$contador"; contador=`expr $contador + 1`; done
 contador=0
 contador=1
 contador=2
 contador=3
 contador=4
 mimbar:/usr/share/linux.see:82>
```

Primeiro atribuímos o valor 0 (zero) à variável *contador*. Depois se a condição *$contador -lt 5* (contador for menor que (**-lt** = **l**ess **t**han) 5), executamos os comandos **echo** e, através do comando **expr**, calculamos um novo valor para a variável contador.

A função test

Existe uma função *test* que permite testar o valor de variáveis e a existência de arquivos. O valor retornado por esta função (*$?*) pode ser testado nos comandos *if/while*.

Vamos ver apenas as formas mais simples:

-f ARQUIVO	verifica se um arquivo existe e não está vazio
-N ARQUIVO	verifica se o arquivo foi modificado desde a última vez que foi lido
ARQ1 -nt ARQ2	se ARQ1 foi modificado mais recentemente que ARQ2
-z VARIAVEL	testa se a variável está vazia
-n VARIAVEL	teste se a variável não está vazia
STRING1 = STRING2	verifica se os dois strings (sequencias de caracteres) são iguais
STRING1 != STRING2	verifica se STRING1 é diferente de STRING2
EXPR1 OP EXPR2	verifica se a expressão 1 (EXPR1) é OP a expressão 2 (EXPR2)

OP é um dos seguintes operadores:

-lt	menor
-le	menor ou igual
-eq	igual
-ne	diferente
-ge	maior ou igual
-gt	maior

Como Linguagem de Programação

Como vamos ver no decorrer deste curso (em particular na seção Estendendo o Ambiente), você pode desenvolver sistemas inteiros utilizando o shell e os diversos utilitários pré-existentes no sistema. Nem sempre o resultado é eficiente mas, como o desenvolvimento é muito rápido, podemos experimentar diversas idéias para resolver os problemas.

Processamento de Textos: Exercícios

Agora que já vimos diversos programas vamos utiliza-los para resolver alguns problemas. Na medida do possível são problemas reais.

Um caminho para resolver os problemas é imaginar como você o resolveria manualmente e depois tentar resolve-lo utilizando o que já vimos. Às vezes, para tratar um problema no computador, devemos encontrar uma representação adequada para ele. A medida que você for adquirindo experiência vai perceber que algumas representações são melhores que outras. Não se acanhe em trocar idéias com seus colegas! Afinal duas cabeças pensam melhor do que uma.

1. Lista de atividades

Você, ao contrário de muita gente, é organizado o suficiente para saber que atividades tem que fazer. Você sabe do livro que deve ler para quinta-feira, a prova que vai ter no

dia 22, etc. E, para aproveitar que tem um computador disponível, resolveu colocar a relação de coisas a fazer em um arquivo. Como você faria isto?

2. Ainda lista de atividades

O sistema que você desenvolveu funciona mas você esta perdendo o controle das coisas. As vezes você concluiu uma tarefa e a removeu do arquivo. Aí você não sabe se ela já foi realizada ou se você esqueceu de coloca-la no arquivo. Seria interessante também conseguir listar as tarefas pendentes ou as tarefas completadas. Como você faria isto? A representação que você utilizou no exercício anterior é boa? Ou você achou uma melhor?

3. Protegendo a lista de atividades

Você está satisfeito com o sistema que desenvolveu. Porém você não quer que todo mundo fique sabendo que, por exemplo, você tem que fazer exame de fezes na sexta-feira. Você consegue evitar que os outros fiquem bisbilhotando sua relação de tarefas?

4. Compartilhando a lista de atividades

Sua lista de atividades funciona muito bem. Tão bem que outras pessoas querem utiliza-la para o trabalho do grupo de física. Você não quer que os outros coloquem mais tarefas para você mas é necessário que eles saibam que você cumpriu as tarefas planejadas. Você consegue resolver isto? Como conciliar o compartilhar da lista de tarefas com o proteger a lista de tarefas?

5. Lista de atividades do grupo

Todos resolveram criar listas das atividades pelas quais são responsáveis. A data de entrega do trabalho esta se aproximando e todo dia você quer saber quais as atividades ainda não foram completadas. Mas você esta ficando cansado de ter que examinar cada lista a cada vez. Como você automatizaria esta tarefa? Você está mais avançado do que seus colegas no uso do computador e gostaria de dar uma mãozinha para eles automatizando o processo de ver as listas de tarefas. Consegue escrever um script para isto?

6. Como o projeto é grande e vai durar o semestre inteiro, seria desejável ao invés de listar sempre toda a lista de atividades listar apenas as novas tarefas e as tarefas completadas. Como você resolveria este problema?

7. Os insatisfeitos...

Todos do grupo ficaram satisfeitos com o que você implementou, porém... Um deles, talvez só para te cutucar, disse: “As minhas atividades eu já sei. Quero que o script só liste as atividades dos outros membros do grupo.”. Você consegue satisfaze-lo?

8. O próximo passo: uma agenda

A idéia da lista de atividades funcionou tão bem e você ficou tão empolgado que resolveu implementar algo mais sofisticado: uma agenda. Qual a grande diferença entre uma agenda e uma lista de atividades? Numa agenda as atividades tem dia e horário para serem realizadas. Como isto muda o que você implementou até agora?

MÓDULO 8

ACESSO A REDE

Nesta parte do treinamento o aluno acessará computadores remotos. Supõe-se que os computadores estejam, pelo menos, ligados em rede. É desejável que eles tenham acesso a Internet.

Quando duas pessoas ou duas máquinas se comunicam é necessário estabelecer entre as partes um protocolo. Um protocolo é um conjunto de regras que deve ser seguido para que uma conversação tenha sucesso. Por exemplo, as duas partes devem falar a mesma língua.

ip

No caso dos computadores, existem diversos protocolos de comunicação. O protocolo mais comum, e que é utilizado pela Internet, é o IP (Internet Protocol).

Este protocolo básico, IP, define como as mensagens são endereçadas e informações sobre o tamanho das mensagens.

O endereçamento é a forma de definir onde a mensagem deve ser entregue. No protocolo IP ele é dividido em duas partes:

- endereço propriamente dito (IP address)
- e o serviço desejado (IP port)

O IP address localiza a máquina na Internet. É como o endereço de uma casa ou apartamento. Este endereço é comumente escrito assim: 64.233.185.106. O IP port discrimina para qual das pessoas da casa é a mensagem (carta ou ligação). Ele é escrito como um número simples (25).

Tanto a telefônica quanto os correios não conhecem os moradores da residência. Conhecem apenas o número do telefone instalado ou o endereço da residência. Mas uma vez que a chamada foi atendida ou a carta recebida ela é encaminhada internamente ao destinatário.

Hoje na Internet (usando a versão 4 do protocolo IP) as máquinas são endereçadas por um número que vai (potencialmente) de 0 (zero) até pouco mais de 4 bilhões (um número de 4 bytes ou 32 bits). Diversos detalhes reduzem o número de máquinas para algo em torno de 100 milhões. Parece muito mas já existem mais de 100 milhões de máquinas no mundo.

Para resolver o problema de endereçamento que já se faz sentir hoje, e que será muito mais crítico no futuro próximo, foi desenvolvida uma nova versão do protocolo IP, chamada IPv6. Além de resolver alguns problemas existentes no encaminhamento das mensagens com o IPv6 poderemos ter 100 endereços IP (máquinas ou outros dispositivos) por metro quadrado habitável do planeta!

A faixa de serviços de cada máquina (IP port) é mais restrito: 65534 (um número de 2 bytes ou 16 bits).

Enquanto as máquinas na Internet são endereçadas através de um número, as pessoas tem dificuldade de lembra-los. Assim utilizamos nomes para as máquinas (www.gmail.com). Um conjunto de máquinas distribuídas pela Internet converte este

nome em endereço IP. Este serviço é o DNS (Domain Name Service — Serviço de Nome de Domínio (máquina)).

Já o número do serviço é descrito em um arquivo na máquina, tipicamente /etc/services. Nele está a informação que o serviço de e-mail (SMTP) utiliza a porta número 25 e o protocolo TCP.

```
mimbar:/usr/share/linux.see:9> grep -i smtp /etc/services
smtp            25/tcp          mail
```

Em cima do protocolo IP novos protocolos podem ser criados.

UDP User Datagram Protocol

TCP Transmission Control Protocol

ICMP Internet Control Message Protocol

Os dois primeiros são mais comuns e utilizados pelas diversas aplicações (DNS, e-mail, web, messenger). O último (ICMP) transporta mensagens de controle que sinalizam o estado da rede, que a máquina destino existe, se o serviço está disponível, se o servidor pode atender a solicitação, etc.

udp

Uma metáfora que ajuda a explicar o protocolo UDP é o das cartas e telegramas. Quando se envia uma carta simples, ela é endereçada, despachada e, normalmente, chega ao destinatário. Você não tem confirmação do recebimento da carta. Duas cartas enviadas no mesmo dia para o mesmo destinatário podem chegar em dias diferentes. Duas cartas enviadas em dias diferentes podem chegar em uma ordem diferente da ordem em que foram enviadas. Devido a estas características este protocolo é não confiável.

Isto não quer dizer que ele não tenha aplicações! Como este protocolo pode-se, por exemplo, enviar várias mensagens sem ter que esperar a resposta. Esta característica é utilizada pelo DNS. O pedido do endereço de uma máquina é enviada a diversos servidores. Utiliza-se a informação do que responder primeiro. Se uma ou mais pedidos ou respostas forem perdidos, ainda assim o protocolo funciona. Desde que um pedido e sua respectiva resposta sejam processados.

tcp

Já o protocolo TCP funciona mais como uma ligação telefônica. Uma pessoa fala e a outra escuta as palavras na mesma ordem em que foram pronunciadas. Existe um protocolo de estabelecimento de conversação (alô) e término (tchau). Se as pessoas forem educadas, uma não tentará falar enquanto a outra está falando (o que originaria ruído para as duas partes e perda de parte da conversação). Se a ligação estiver ruim ou ruidosa, uma parte sempre pode pedir retransmissão do trecho que não entendeu. Como há garantias que as mensagens são entregues corretamente, este protocolo é chamado de confiável.

Novos protocolos, mais específicos, podem ser criados utilizando os protocolos UDP e TCP. Vamos examinar rapidamente três deles.

Logo que a Internet foi criada, as pessoas ainda estavam explorando diversas idéias. Era muito mais importante conseguir escrever e testar os programas e protocolos do

que faze-los eficientes. Assim, muitos dos protocolos são textos curtos, normalmente uma ou duas palavras em inglês. Seguido, às vezes, por mais informações ou comentários.

Uma exceção é o DNS. O protocolo utilizado é binário. Um dos motivos desta decisão foi a utilização do protocolo UDP que tem um limite máximo para o tamanho das mensagens. Mais recentemente foi percebido que esta não foi uma boa decisão mas, por questões de compatibilidade, não será alterada.

Uma forma de utilizar, experimentar e aprender estes protocolos é utilizando o programa **telnet**. Ele funciona como um terminal remoto para o serviço desejado. Possui dois argumentos:

1. o nome ou endereço IP do servidor

2. o nome do serviço ou o número da porta desejada

Para se encerrar uma sessão de telnet digite '^]' (um único caractere: segure a tecla control e pressione ']') e, em seguida 'q'.

daytime

Este é um dos protocolos mais simples que existe. Acessando o serviço de daytime (porta 13) o servidor retorna a data e hora correntes. Nenhuma interação é necessária.

```
mimbar:/usr/share/linux.see:3> telnet time.nist.gov daytime
Trying 192.43.244.18...
Connected to time.nist.gov.
Escape character is '^]'.
53647 05-10-04 01:57:39 27 0 0 491.1 UTC(NIST) *
Connection closed by foreign host.
```

A parte que interessa do resultado é 05-10-04 01:57:39. Isto é a data no formato ano-mês-dia e o horário no formato hora-minuto-segundo. Como este serviço pode ser acessado de qualquer ponto da Internet, o horário retornado é o UTC, antigo horário padrão de Greenwich. Fora do horário de verão, Brasília está 3 horas atrasada em relação a Greenwich.

http

Este protocolo (HyperText Transfer Protocol) é utilizado para acessar as páginas web (WWW — World Wide Web). Ele utiliza TCP e o serviço está na porta 80.

```
mimbar:~:8> telnet bonehunter.rulez.org http
Trying 193.225.158.7...
Connected to gandalph.mad.hu.
Escape character is '^]'.
(1) HEAD / HTTP/1.0
(2)
(3) HTTP/1.0 404 Not Found
(4) Date: Mon, 03 Oct 2005 22:52:59 GMT
(5) Connection: close
(6) Content-type: text/html
(7) Server: Thy/0.9.4 Debian (i386) GnuTLS/1.0.16 zlib/1.2.2 PHP/
4.3.10-16 (Linux)
(8)
(9) Connection closed by foreign host.
```

Para fins de referência, numerei as linhas com (n). Estes números não foram digitados nem recebidos.

As linhas (1) e (2) são a requisição ao servidor web. No caso simplesmente foi solicitada informação sobre a página de entrada (/) do site (bonehunter.rulez.org). A linha (2), em branco, indica fim de requisição.

As linhas (3) a (8) são a resposta do servidor. O código 404 na linha (3) indica que a página não foi encontrada. A linha (4) indica o horário da resposta. A (5) que a conexão será fechada. A (6) indica que o resultado foi texto (text/html). Poderia ter sido uma imagem (image/gif), ou um filme (image/mpeg), ou um documento Microsoft Word (text/msword), etc. Na linha (7) o servidor se identifica e a linha vazia (8) indica fim de cabeçalho da resposta. Se houvesse mais dados, uma imagem, por exemplo, ele viria após a linha (8). A linha (9) indica que o servidor fechou a conexão.

smtp

O protocolo SMTP (Simple Mail Transfer Protocol) é utilizado para a transferência de e-mails através de TCP/IP (existem outros métodos que estão quase extintos). A porta associada a este serviço é 25.

Ao contrário dos protocolos acima, este é composto de uma série de comandos e respostas onde o próximo comando depende das respostas anteriores. Como dito acima, este protocolo utiliza textos em inglês, tanto para os comandos como para as respostas. Para facilitar o processamento por programas, o primeiro campo nas respostas é um número. O resto da linha é um comentário para os humanos envolvidos no processo.

```
 mimbar:/usr/share/linux.see:7> telnet mx1.hotmail.com smtp
 Trying 64.4.50.99...
 Connected to mc1.bay6.hotmail.com.
 Escape character is '^]'.
 220 mc1-f9.hotmail.com Sending unsolicited commercial or bulk e-
mail to Microsoft's computer network is prohibited. Other
estrictions are found at http://privacy.msn.com/Anti-spam/.
Violations will result in use of equipment located in California
and other states. Tue, 4 Oct 2005 07:46:14 -0700
 HELO localhost
 250 mc1-f9.hotmail.com (3.0.1.19) Hello [201.19.144.18]
 MAIL FROM:<nobody@example.com>
 250 nobody@example.com....Sender OK
 RCPT TO:<nobody@hotmail.com>
 250 nobody@hotmail.com
 DATA
 354 Start mail input; end with <CRLF>.<CRLF>
 Subject: E-mail de exemplo
 O corpo comeca apos a primeira linha em branco.
 .
 250 <MC1-F976UaiRNhFCfyw0096a54d@mc1-f9.hotmail.com> Queued mail for
delivery
 quit
 221 mc1-f9.hotmail.com Service closing transmission channel
 Connection closed by foreign host.
```

As linhas que começam com números (220, 250, 354, 250, 221) são as respostas do servidor. As outras são os comandos enviados.

1. Ao se conectar ao servidor, ele responde se identificando (mc1-f9.hotmail.com).

2. Nos identificamos junto ao servidor (HELO localhost).

3. Ele responde que está pronto para receber os dados para endereçamento da mensagem (250).

4. Informamos quem é o remetente (MAIL FROM:<nobody@example.com).

5. O servidor informa que o remetente foi aceito (250).

6. Informamos o destinatário da mensagem (RCPT TO:<nobody@hotmail.com).

7. O destinatário é aceito (250).

8. Agora que o "envelope" da mensagem foi preenchido passamos a enviar a mensagem propriamente dita (DATA).

9. O servidor diz que está pronto para recebe-la (250) e que ela deve terminar com uma linha contendo apenas um ponto '.'.

10. Enviamos o campo de assunto da mensagem (Subject:) e uma linha em branco para informar que o cabeçalho da mensagem já terminou.

Em seguida escrevemos o corpo da mensagem e a finalizamos com um '.'.

11. O servidor informa que a mensagem foi aceita (250), armazenada e que vai ser (eventualmente) entregue.

12. Como não temos mais mensagens a enviar, finalizamos a conversação (QUIT).

13. O servidor comunica o fim de transmissão (250) e fecha a conexão.

Enquanto o telnet é uma ótima ferramenta para o acesso interativo aos diversos serviços disponíveis na rede, ele não se presta a utilização não-interativa, como em um shell-script. Para isto existe uma outra ferramenta o netcat. Esta ferramenta funciona como o cat porém, ao invés de escrever em um arquivo local, ele envia os dados, via conexão TCP, para um servidor. Ao mesmo tempo, ele pega as informações enviadas do servidor e as escreve na saída padrão. Veja o exemplo abaixo:

```
 mimbar:/usr/share/linux.see:16> echo -e "HEAD / HTTP/1.0\n" |
netcat slashdot.org http
 HTTP/1.0 200 OK
 Date: Sun, 06 Nov 2005 12:21:15 GMT
 Server: Apache/1.3.33 (Unix) mod_gzip/1.3.26.1a mod_perl/1.29
 SLASH_LOG_DATA: shtml
 X-Powered-By: Slash 2.005000087
 X-Bender: The laws of science be a harsh mistress.
 Cache-Control: private
 Pragma: private
 Content-Type: text/html; charset=iso-8859-1
 Connection: close
```

Os seus resultados podem ser um pouco diferentes devido a configuração do seu ambiente.

Acesso a Rede: Exercícios

Metáforas:

UDP cartas e telegramas (fora de ordem e sem confirmação)

TCP telefone (seqüencial e bidirecional)

Acesso a Rede: Exercícios

Vamos a alguns exercícios utilizando a rede.

1. No último exemplo, você pode observar uma linha que começa com X-Bender ou X-Fry. Estas linhas contém citações de dois personagens da série de animação Futurama. Como você poderia imprimir apenas estas linhas? Utilize o comando netcat descrito em Acesso a Rede.

MÓDULO 9

ESTENDENDO O AMBIENTE

Apesar do extenso conjunto de utilitários e ferramentas disponíveis no unix, nem sempre conseguimos resolver nossos problemas com elas. Ou queremos um pouco mais de funcionalidade da ferramenta. Neste caso precisamos escrever novas ferramentas. Quando as novas ferramentas são implementadas como filtros elas podem ser facilmente reutilizadas.

As ferramentas podem ser escritas em uma variedade de linguagens:

- **C** – linguagem compilada de alta performance
- **shell** – e por que nao?
- **perl** – linguagem interpretada voltada ao processamento de textos
- **tcl** – linguagem interpretada voltada a composição de ferramentas
- **[ruby]** – linguagem interpretada orientada a objetos
- **[python]** – linguagem interpretada de alto nível

A diferença entre uma linguagem interpretada e uma linguagem compilada é que as linguagens compiladas são executadas diretamente pelo processador enquando uma linguagem interpretada é executada por um programa, que por sua vez foi compilado. Existem mais detalhes mas isto é o suficiente por enquanto.

Um novo rm

Como vimos em rm, uma vez que um arquivo foi removido ele não pode ser recuperado. Seria interessante alterar o comando rm de forma a preservar uma cópia do arquivo para o caso de mudarmos de idéia. Isto não elimina a necessidade de backups!

Uma forma de fazermos isto é criar um novo programa rm e coloca-lo no PATH antes da versão do sistema. Para tal alteramos a variável $PATH para incluir o diretório onde está a nossa versão do rm antes do diretório /bin e/ou /usr/bin. Um bom diretório para colocar estas versão incrementadas dos programas é em $HOME/bin.

Os seguintes passos criam um novo ambiente com um rm seguro:

```
cd $HOME
mkdir bin BACKUP
PATH=$HOME/bin:$PATH
export PATH
cd bin
cat >rm
#!/bin/sh
/bin/mv $* $HOME/BACKUP
^D
chmod 755 rm
```

Anagramas

Anagramas são palavras que possuem as mesmas letras mas em ordem diferentes. Um exemplo clássico são

- ROMA
- AMOR
- RAMO
- MORA (de morar)
- ARMO (de armar, engatilhar)
- ORAM (de orar)
- ROAM (de roer)

Pare um pouco e pense como você acharia estas palavras manualmente.

Pense um pouco mais. Quais características estas palavras tem em comum?

Com as ferramentas que você tem até aqui você consegue resolver o problema?

Será que existe uma expressão regular que possa descrever estas palavras? Num certo sentido a resposta é não. O que você poderia escrever seria uma expressão regular com todas as combinações possíveis das quatro letras ([AMOR]) e procura-las no dicionário selecionando as que são palavras válidas.

Se você pudesse criar um arquivo desta forma:

AMOR ROMA

AMOR AMOR

AMOR MORA

AMOR ARMO

AMOR ORAM

AMOR RAMO

AMOR ROAM

onde o primeiro campo são as letras em ordem alfabética e o segundo campo a palavra seria fácil, usando as ferramentas já conhecidas, encontrar os anagramas. O primeiro campo seria a assinatura da palavra.

Vejamos como seria o programa ou script para achar a assinatura de uma palavra. A palavra é passada como argumento.

Em shell: assinatura.sh

```
#!/bin/sh
# Primeiro colocamos cada letra em uma linha
# Depois ordenamos o arquivo
# Em seguida recolocamos tudo em uma linha so
# Entao tiramos os espacos em branco
# Finalmente imprimimos a palavra
echo ‘
echo $1 | \
sed -e ‘s/./&\n/g’ | \
sort‘ | \
tr -d ‘ ‘
```

Testando:

```
mimbar:/usr/share/linux.see:12> /assinatura.sh roma
amor
mimbar:/usr/share/linux.see:13> ./assinatura.sh sucesso
ceosssu
```

Algumas coisas que não fizemos foram:

- converter a palavra em minúsculas ou maiúsculas
- imprimir a palavra original

Seguindo a filosofia dos filtros, um programa deve fazer uma única função e faze-la bem feito. Isto possibilita maior flexibilidade.

Em perl: assinatura.pl

```
#!/usr/bin/perl -w
print sort(split("", shift)), "\n";
```

Testando:

```
mimbar:/usr/share/linux.see:20> /assinatura.pl nosso
nooss
mimbar:/usr/share/linux.see:21> ./assinatura.pl sucesso
ceosssu
```

O script não foi escrito de uma forma didática mas sim como ele seria na vida real. Para entende-lo precisamos ir do centro para fora:

shift retorna o primeiro argumento

split com o primeiro argumento vazio ("") separa as letras

sort ordena a lista de letras

print imprime o resultado (precisamos do "\n" para terminar a linha)

Em tcl: assinatura.tcl

```
#!/usr/bin/tclsh
puts [join [lsort [split [lindex $argv 0] ""]] ""]
```

Testando:

```
mimbar:/usr/share/linux.see:27> /assinatura.tcl rodeio
deioor
mimbar:/usr/share/linux.see:28> ./assinatura.tcl sucesso
ceosssu
```

Novamente o script não é didático. O que ele faz é:

lindex $argv 0 pega o primeiro argumento

split separa em letras

sort ordena a lista

join junta as letras

puts imprime

No diretório /usr/share/linux.see existe um arquivo, pequeno.dicionário, com algumas centenas de palavras (nem todas são válidas). Você pode fazer algumas experiências com ele. Trabalhando apenas com palavras de quatro letras achei:

```
cama
maca
alma
lama
mala
lata
tala
isca
saci
caju
jacu
asco
caos
caso
saco
```

E a maior seqüencia que achei foi:

```
alentado
enlatado
entalado
toledana
tonelada
```

Exercicios

• Usando uma das versões do programa de assinaturas, crie um script para achar os anagramas de 7 letras no arquivo pequeno.dicionario.

MÓDULO 10

O SISTEMA DE ARQUIVOS COMO BASE DE DADOS

Nesta sessão veremos como utilizar o que foi visto anteriormente para armazenar, organizar e recuperar dados em um sistema Unix. Como os programas utilizados já foram vistos, teremos poucos exemplos e mais exercícios. Considere esta sessão uma "Sessão Avançada".

Base de Dados versus Banco de Dados

Através de uma representação adequada o sistema de arquivos do Unix pode ser utilizado como uma base de dados.

Note que ele pode implementar uma base de dados e não um banco de dados. A diferença pode parecer sutil mas os bancos de dados implementam funcionalidades que o sistema de arquivos por si só não é capaz. Esta funcionalidade é abreviada como ACID:

A **Atomicidade** — ou a operação acontece ou não. Resultados intermediário não são perceptiveis.

C **Consistência** — não é possível executar uma operação que gere dados inválidos

I **Isolamento** — se duas ou mais operações acontecem ao mesmo tempo, o resultado é o mesmo se elas tivessem sido executadas uma após a outra. Em particular, uma operação não vê os resultados intermediários da outra.

D **Durabilidade** — uma vez terminada uma operação os seus resultados são permanentes no sentido que persistem mesmo que o computador falhe.

Se o sistema de arquivos não consegue suprir as funções acima ele ainda pode ser utilizado como base de dados? A resposta é afirmativa. Nem sempre as funções acima são importantes para a aplicação. E um sistema de banco de dados pode ser caro em recursos computacionais ou financeiros.

Vamos criar uma base de dados e trabalhar com ela. No processo vamos utilizar os programas que vimos nas seções anteriores. Usaremos receitas diversas como dados. Estas receitas foram obtidas de diversas fontes.

Criação de uma Base de Dados

O primeiro passo para armazenar os dados, seja num computador ou em um fichário, é descobrir quais informações são importantes e escolher uma forma conveniente para representa-las. É melhor criar um livro de receitas com todas as receitas em um arquivo ou um arquivo para cada receita?

Pelo que vimos até agora, um arquivo texto por receita é bem conveniente.

Resolvido este ponto vamos a representação dos dados dentro do arquivo. Podemos dizer que cada receita (arquivo) possui alguns atributos como nome, autor, ingrediente, etc.

1. Podemos ter apenas um texto descrevendo a receita. Esta representação é muito versátil mas torna difícil recuperar as informações depois.

2. Podemos colocar todos os ingredientes na primeira linha, por exemplo, e as instruções de preparo na segunda, o autor na terceira, o rendimento na quarta, etc. (podemos fazer isto porque as linhas não tem limite de tamanho).

3. Podemos criar seções com as partes importantes da receita (ingredientes, preparo,

montagem, etc.). É interessante porque teremos todas as informações juntas mas dificulta utilizar as ferramentas do unix que trabalham por linha.

4. Podemos colocar uma marca em cada linha para identificar o tipo da linha, se a linha descreve um ingrediente, um tempo, um autor, etc. Isto ocupa mais espaço, que hoje é bem mais barato que anteriormente, mas ganhamos flexibilidade.

Aqui vamos adotar a última opção.

Feito isto precisamos definir a ordem em que as diversas informações aparecem dentro da receita. Primeiro o nome da receita, depois o autor, depois o rendimento, depois o custo, etc. Mas e se uma informação não existir, por exemplo, o autor? E se quisermos acrescentar mais informações? Vamos coloca-las no início, meio ou fim?

A representação com uma marca no início de cada linha resolve este tipo de problema.

Resolvidos estes problemas vamos por a mão na massa :-) Veja como ficou uma pequena receita:

```
mimbar:/usr/share/linux.see/Receitas:47> cat Pizza_Caprina.txt
Nome: Pizza Caprina
Autor: Saul Galvão
Ingrediente: 450g de massa para pizza
Ingrediente: 250g de mussarela especial em fatias
Ingrediente: 120g de queijo de cabra tipo Chevrotin em fatias
Ingrediente: 08 folhas de sálvia
Preparo: Abrir a massa para pizza.
Preparo: Distribuir a mussarela especial e o queijo de cabra.
Preparo: Levar ao forno.
Preparo: Retirar e colocar 1 folha de sálvia sobre cada pedaço.
```

Acesso a Base de Dados

Uma base de dados não é muito útil se não conseguimos extrair os dados dela. Uma forma simples é utilizar o nome do arquivo para achar uma receita. Vejamos o que temos por aqui:

```
mimbar:/usr/share/linux.see/Receitas:49> ls
Abobora_Suica.txt                          Pasta_de_atum_fresco.txt
Chutney_de_Figos.txt                       Piperade.txt
Chutney_de_Tomates.txt                     Pizza_Caprina.txt
Cocada.txt                                 Pudim_de_abacaxi.txt
Creme_de_banana_e_laranja.txt              Pudim_de_maria mole_enriquecido.txt
Espetinhos_Tropicais.txt                   Quibe.txt
File_ao_Molho_Mostarda.txt                 Rolinho_de_Siri.txt
Frango_Empanado_com_Queijo_Parmesao.txt    Salada_Alema.txt
Frango_de_Mineiro_na_Praia.txt             Salada_Dinamarquesa.txt
Gelado_de_Morango.txt                      Salada_colorida.txt
Homus.txt                                  Salada_da_Copa.txt
Lasanha_aos_Quatro_Queijos.txt
Salada_de_Macarrao_com_Brocolis.txt
Lombinho_de_Porco_Grelhado.txt             Salada_de_Papaia.txt
Macas_Assadas.txt                          Sanduiche_Pizza.txt
Manjar_de_coco_com_calda_de_amora.txt      Sorvete.txt
Misto_Quente_com_Cobertura.txt             Torta_de_Fuba.txt
Omelete_Basica.txt
```

A relação de arquivos na sua máquina pode ser diferente da relação acima!

Os nomes precisam ser sugestivos para encontrarmos as receitas. E precisam ser únicos também. Ou seja, você não pode ter duas receitas de sorvete em arquivos com o nome Sorvete.txt E, quando a lista ficar com centenas de nomes, vai ser preciso muita paciência para procurar as receitas!

Exercícios

- Qual a sua proposta para tratar receitas com o mesmo nome?
- Como você resolveria o problema de tratar um grande número de receitas? Como você as agruparia?
- Qual o impacto da sua solução acima para achar uma determinada receita?
- Qual seria a regra de formação dos nomes dos arquivos de receitas?

Pesquisando

Seria interessante pesquisar pelos atributos das receitas. Por exemplo, poderiamos listar apenas as sobremesas :-)

Vamos começar com algo mais simples, que servirá de modelo para os próximos refinamentos. Vamos listar os títulos das receitas:

```
 mimbar:/usr/share/linux.see/Receitas:12> grep ^Nome: *
 Abobora_Suica.txt:Nome: Abóbora suíça
 Chutney_de_Figos.txt:Nome: Chutney de figos
 Chutney_de_Tomates.txt:Nome: Chutney de Tomates
 Cocada.txt:Nome: Cocada
 Creme_de_banana_e_laranja.txt:Nome: Creme de banana e laranja
 Espetinhos_Tropicais.txt:Nome: Espetinhos Tropicais
 File_ao_Molho_Mostarda.txt:Nome: Filet ao Molho Mostarda do Mar-
celo
 Frango_de_Mineiro_na_Praia.txt:Nome: Frango de mineiro na praia
 Frango_Empanado_com_Queijo_Parmesao.txt:Nome: Frango empanado com
queijo parmesão
 Gelado_de_Morango.txt:Nome: Gelado de Morango
 Homus.txt:Nome: Homus
 Lasanha_aos_Quatro_Queijos.txt:Nome: Lasanha aos Quatro Queijos
 Lombinho_de_Porco_Grelhado.txt:Nome: Lombinho e Porco Grelhado
 Macas_Assadas.txt:Nome: Maçãs Assadas
 Manjar_de_coco_com_calda_de_amora.txt:Nome: Manjar de coco com
calda de amora
 Misto_Quente_com_Cobertura.txt:Nome: Misto quente com cobertura
 Murdadaras.txt:Nome: Arroz com Lentilhas
 Murdadaras.txt~:Nome: Arroz com Lentilhas
 Omelete_Basica.txt:Nome: Omelete Básica
 Pasta_de_atum_fresco.txt:Nome: Pasta de atum fresco
 Piperade.txt:Nome: Piperade
 Pizza_Caprina.txt:Nome: Pizza Caprina
 Pudim_de_abacaxi.txt:Nome: Pudim de abacaxi
 Pudim_de_maria-mole_enriquecido.txt:Nome: Pudim de maria-mole
enriquecido
 Rolinho_de_Siri.txt:Nome: Rolinhos de Siri
 Salada_Alema.txt:Nome: Salada alemã
 Salada_colorida.txt:Nome: Salada colorida
```

```
 Salada_da_Copa.txt:Nome: Salada da Copa
 Salada_de_Macarrao_com_Brocolis.txt:Nome: Salada de macarrão com
brócolis
 Salada_de_Papaia.txt:Nome: Salada de papaia, manga e camarão
 Salada_Dinamarquesa.txt:Nome: Salada Dinamarquesa
 Sanduiche_Pizza.txt:Nome: Sanduíche pizza
 Sorvete.txt:Nome: Gelado de Morango
 Torta_de_Fuba.txt:Nome: Torta de Fubá
```

Observe que, devido ao formato flexível das informações que adotamos, nenhum atributo é obrigatório!

Será que encontramos todos os títulos de receitas?

```
 mimbar:/usr/share/linux.see/Receitas:13> ls -1 | wc
-l135
 mimbar:/usr/share/linux.see/Receitas:14> grep ^Nome: * | wc
-l134
```

Oops. Parece que não.

Exercícios

- Liste os arquivos que não possuem o campo Nome:.
- Liste apenas os títulos das receitas, sem o nome do arquivo onde elas estão. Dica: existem, pelo menos, duas formas de fazer isto. Uma passando uma opção para o comando grep e outra processando a saída do comando grep.

Os Próximos Passos

Chegamos ao final do curso de Introdução ao Linux. Espero que tenham aproveitado e percebido que utilizar o Linux e programar não é tão difícil assim.

No início da apostila, em **Onde Podemos Chegar?**, vimos um programa para pesquisa de receitas. Abaixo está o código do programa, na realidade um *shell script*. Este script foi escrito para ser simples e didático e usa o que foi exposto ou um pouco mais.

Shell Script: Receitas

```
#!/bin/sh
# script para pesquisa de receitas
DB="/usr/share/linux.see/pool"
OP="$1"
OQUE="$2"
unset CDPATH
cd $DB
if test "$OP" = "com"
then
     # receitas com ingrediente
     LISTA='grep -l -i "^Ingrediente: .*$OQUE" ???'
     if test -z "$LISTA"
     then
     echo "Ingrediente '$OQUE' nao encontrado em nenhuma receita!" 1>&2
        exit 1
     fi
     for qual in $LISTA
     do
        titulo='grep "^Nome: " $qual | cut -d" " -f2-'
        echo "$qual: $titulo"
     done
elif test "$OP" = "de"
then
     # receitas de autor
     LISTA='grep -l -i "^Autor: .*$OQUE" ???'
     if test -z "$LISTA"
     then
          echo "Autor '$OQUE' nao encontrado em nenhuma receita!" 1>&2
        exit 1
     fi
     for qual in $LISTA
     do
        titulo='grep "^Nome: " $qual | cut -d" " -f2-'
        echo "$qual: $titulo"
     done
elif test "$OP" = "lista"
then
     # receita lista numero
     if test -f $OQUE
     then
        cat $OQUE
```

```
    else
        echo "$0: Nao existe a receita $OQUE"
       exit 1
    fi
else
    # opcao invalida
    echo "$0: use uma das opcoes abaixo"
    echo "$0 com ingrediente"
    echo "$0 de autor"
     echo "$0 lista numero-da-receita"
    exit 1
fi
exit 0
```

E agora?

Que tal fazer uma pequena alteração e aceitar pesquisa por tipo de receita? Crie a opção de pesquisar as receitas de sobremesa, por exemplo.

GLOSSÁRIO

compilador – um conjunto de programas que transforma o código fonte em um código objeto que é diretamente executado pela máquina.

interpretador – um conjunto de programas que transforma o código fonte em um formato intermediário que é então executado.

banco de dados – conjunto de dados relacionados e suas operações.

backup – processo de resguardar um conjunto de arquivos de forma a resistir a falhas tanto em hardware quanto em software. Também conhecido como cópia de segurança

bit – a menor quantidade de informação, usualmente representado por 0 (zero) ou 1 (um).

byte – um conjunto de (usualmente) 8 bits. É capaz de representar 256 valores diferentes.

Free Software Foundation – entidade de fomento do software livre e, em particular, ao Projeto GNU.

GNU – Gnu Not Unix. Uma sigla recursiva de um projeto com o objetivo de criar um ambiente unix livre de amarrações proprietárias

hash – literalmente quer dizer picadinho, como em picadinho de carne. Em computação significa uma transformação de um dado em uma forma com menos forma/estrutura.

HD – ver winchester

md5 – Message Digest #5 — função hash para teste de integridade de mensagem

TCL – abreviatura de Tool Command Language (linguagem para controle de ferramentas)

sha1 – Secure Hash Algorithm revisão #1

Grupo de usuários – Quando diversos usuários compartilham arquivos é interessante colocar os usuários em um grupo para facilitar o acesso aos arquivos.

winchester – dispositivo de armazenamento magnético. Também conhecido como disco rígido, hard drive ou HD.

Red Hat - uma das mais antigas distribuições linux. Serve como padrão ou referência a diversas outras. Origem: Estados Unidos.

SuSE - o ênfase desta distribuição é o conjunto das aplicações que foram configuradas para rodarem bem umas com as outras. Origem: Alemanha.

Debian - uma distribuição 100% livre. É a que possui mais pacotes e roda em computadores onde o Microsoft Windows não roda. Desenvolvida por um grande conjunto de voluntários espalhados pelo mundo.

Knoppix - uma distribuição baseada no Debian que roda toda de um CD ou DVD. Você pode utiliza-la sem ter que formatar o winchester da sua máquina.

Ubuntu - outra distribuição baseada no Debian mas de instalação e administração mais fácil. Saem duas versões por ano.

Turbolinux - uma distribuição para a Ásia.

Mandriva/Conectiva/Mandrake - a Conectiva era uma distribuição brasileira de linux, e uma das primeiras em portugues. Foi unida a Mandrake, que é francesa, e se tornaram

Mandriva.

Metasys - uma distribuição baseada em Fedora Core (um tipo de Red Hat) em portugues. O ênfase é a facilidade de atualização e a lingua.

root - usuário todo poderoso (super-usuário) e administrador do sistema. Ele pode criar e remover outros usuários, ler e escrever em qualquer arquivo.

montar/montagem - quando um disco ou outro meio de armazenamento é acrescentado à estrutura de diretório é dito que o disco foi "montado" no sistema de arquivos. É parecido com montar uma roda em um carro depois de consertar o pneu.

mimbar - planeta natal dos Mimbari, do seriado de ficção científica Babylon 5.

bit - diminutivo de BInary digitT, é a menor quantidade de informação, capaz de representar apenas sim/não, ligado/desligado, presente/ausente.

byte - um conjunto de (tradicionamente) 8 bits. Um byte corresponde grosseiramente a um caractere.

word/palavra - um conjunto de bytes que é manipulado como uma unidade pelo processador (CPU).