BIBLIOTECA BÁSICA

INFORMÁTICA

UNIX 31

o futuro dos sistemas operacionais

SÉCULO FUTURO

BIBLIOTECA BÁSICA

INFORMÁTICA

UNIX 31

o futuro dos sistemas operacionais

SECULO FUTURO

Diretor editor:
M. A. Nieto

Coordenação e supervisão técnica:
Engº Sergio Rocha Paggioli

Tradução:
Ideli Novo

Projetos especiais:
Rainer K.E. Ladewig

Editor Gráfico:
Auro Pereira da Silva

Revisão:
Susana M. Amaral Couto

Produção Gráfica:
Luiz Carlos Siqueira Lago

Arte-Final:
Rubens Tadeu Benedito (coordenador)
Luiz Antonio de Andrade
Vadinho de Oliveira

Fotocomposição, fotolito:
Omnicolor Gráfica e Propaganda Ltda - Rua Dr. Virgílio de Carvalho Pinto, 619
Pinheiros - CEP 05415 - São Paulo.

Impressão:
Editora Antártica S.A. - Av. Ramon Freire, 6920 (Pajaritos) - Santiago - Chile

ÍNDICE

CAPÍTULO VI

APÊNDICE

PREFÁCIO

UNIX são as siglas de identificação de um dos melhores sistemas operacionais de aplicação geral que se encontra no mercado de micro e mini-computadores neste momento. E dissemos um dos melhores para não dizer o melhor. Nós explicaremos: UNIX é, antes de tudo, um sistema operacional de uso geral, isto é, que possui ferramentas e utilitários capazes de efetuar aplicações de gestão, controle industrial, comunicações, cálculo científico, desenho gráfico, etc. UNIX não está limitado a uma aplicação concreta.

Um dos objetivos fundamentais que se propuseram os criadores do UNIX foi sua transportabilidade e facilidade de implementação em diversos tipos de hardware. Isto foi conseguido através da criação de uma linguagem de programação denominada C, e a escrita do núcleo do sistema UNIX nesta linguagem, com o que deve-se, unicamente, possuir um compilador de C na máquina onde se deseja implementar o sistema operacional UNIX. Os controladores de dispositivos (discos, fitas, terminais, impressoras, etc.) eram escritos em código de máquina ou então em Assembler, com o qual deviam ser reescritos para cada tipo de hardware. Atualmente, quase todos estão escritos em linguagem C e para numerosos tipos de hardware que possa suportar UNIX, reafirmando assim, a sua transportabilidade de uma máquina à outra.

A linguagem C está intimamente ligada ao sistema operacional UNIX e praticamente não se pode entender um sem o outro.

Dado que tal linguagem possui uma sintaxe de alto nível, tem potência para manejar o sistema a baixíssimos níveis. Ao estar profundamente inter-relacionado com todo UNIX, conseguiu-se um sistema operacional com potência para poder desenvolver qualquer tipo de utilidade. UNIX possui grande quantidade de comandos e programas complexos, porém oferece possibilidades de programação, controle de processamentos, multitarefa e controle de fluxos, que o fazem ideal para qualquer tipo de aplicação, não estando limitado, como muitos sistemas operacionais, a empenhar-se em um tipo concreto de aplicação.

Vejamos dois aspectos praticamente contraditórios do UNIX: por um lado, é um sistema operacional com poucos conceitos básicos, muito claros e bastante fáceis de entender, os quais uma vez dominados nos oferecem possibilidades imensas de trabalho; por outro, tem-se a impressão de que é um sistema operacional demasiadamente complexo e que "**falha**" em algumas ocasiões. Efetivamente, é um sistema complexo, como dissemos antes, porém em absoluto com falhas monumentais face ao usuário. Existiram e existem numerosas versões do UNIX, que vem melhorando com o tempo; nunca se poderá dizer do UNIX, nem de nenhum outro sistema operacional, que está criada a versão definitiva e que não se possa melhorá-lo. Agora sim, as falhas do UNIX são de responsabilidade total do usuário e da falta de conhecimento geral que existe sobre este sistema operacional. Pode-se dizer que UNIX nunca pede confirmação para executar o comando que lhe foi enviado e, por este motivo, pode-se produzir casos de alterações importantes da informação, mas tudo isto é devido à falta de preparação dos usuários. Talvez possamos pedir ao UNIX algo mais de controle nestes aspectos, porém não podemos desvirtuar um sistema como UNIX por um detalhe desse tipo. Também falta uma boa documentação para o usuário, e inclusive os manuais originais do UNIX são demasiadamente extensos e pouco claros (críticos).

UNIX possui uma extensa biblioteca de programas especiais para diversas tarefas e cada vez existem mais no mercado. É um sistema que tem assentadas as bases de sua estrutura e funcionamento. Será melhorado, cremos, com o incremento da potência do hardware que incorporem as máquinas ou computadores. Talvez a maior tarefa que resta ao UNIX seja o aprendizado dos usuários finais do mesmo, e inclusive do pessoal técnico que trabalha com o UNIX. Existe um numeroso conglomerado de livros e publicações sobre o tema, porém muitos não fazem mais do que contribuir à confusão geral. Nossa opinião como usuários do UNIX e,

ainda hoje, estudiosos do mesmo, é que se devem enfatizar muito as idéias e estruturas básicas do UNIX durante o tempo em que seja necessário, e não continuar com o resto do sistema até o domínio total destes conceitos. Uma vez obtida a confiança para poder seguir adiante, continuaremos navegando pelo extenso caminho do UNIX, recorrendo sempre, mas em último caso, aos manuais originais.

Por tudo isto, o livro que apresentamos a seguir não pretende ser nem uma substituição dos manuais, nem uma pequena iniciação ao sistema. Pretendemos reforçar e dar exemplos dos conceitos básicos do UNIX de tal forma que o usuário possa adquirir a confiança necessária para continuar o estudo em outros níveis. A clareza e, também, a complexidade do UNIX e da linguagem C nos levou à idéia de criar uma obra do UNIX que, também, sirva como introdução ao sistema e aprofundar-se em numerosas partes que, se não forem tratadas desde o início, podem chegar depois a ser totalmente obscuras para o usuário final. Quando o usuário continua adiante com o sistema sem essa forte base nos conceitos UNIX, poderá dizer que o sistema "**falha**". Devemos notar que seria muito conveniente unir a leitura do presente livro com o da Linguagem C (número 27 da Biblioteca Básica Informática), pois, como dissemos anteriormente, não se pode entender um sem o outro.

O livro começa com uma narração histórica do UNIX. Na seqüência explicaremos o que o usuário vai encontrar quando entra no sistema. Continuaremos com o desenvolvimento dos conceitos básicos sobre arquivos UNIX e os comandos para manejá-los, que é o ítem mais extenso do livro. Mais tarde nos adentraremos em um comando básico, o editor de arquivos **(ed)**. Avançaremos no estudo do restante dos conceitos fundamentais da Shell do sistema, finalizando com algumas noções sobre a estrutura interna do núcleo do sistema. Acrescentaremos uma tabela completa de comandos UNIX (excetuando aqueles que ultrapassem o nível de um usuário médio), com todas suas opções possíveis; esta tabela servirá de guia para o usuário uma vez dominados o restante dos capítulos do livro.

Como todo livro de desenvolvimento de um sistema operacional, convém lê-lo mais de uma vez, pela simples razão de que a complexidade de um sistema operacional, não somente do UNIX, requer o manejo de tantos conceitos e definições que é impossível seguir uma ordem totalmente coerente no desenvolvimento dos temas. Assim podemos encontrar no princípio do livro conceitos que são amplamente desenvolvidos posteriormente e que,

em uma segunda leitura, nos dariam uma visão muito mais completa e clara do tema tratado.

Esperamos com isto poder assentar as bases que necessita o mero interessado ou o bom usuário e posterior programador UNIX. Não pretendemos fazer uma obra extensa e complexa, que não levaria mais que ao esquecimento de todo o estudo, mais sim uma potente introdução a este sistema operacional. Esperamos havê-lo conseguido.

CAPÍTULO I

DESENVOLVIMENTO HISTÓRICO DO UNIX

Necessidade do UNIX

O UNIX é um Sistema Operacional que foi desenvolvido no início da década de setenta pelo pessoal da "BELL TELEPHONE LABORATORIES" (NEW JERSEY-USA), pertencentes à empresa AT&T.

Estes laboratórios se estabeleceram em 1925 e podem ser considerados como um dos mais importantes grupos de pesquisa do mundo. Definitivamente, dedicam-se a realizar pesquisas dentro do mundo científico, como por exemplo, as comunicações, a matemática, a física e, de maneira especial, a informática (desenho auxiliado por computador, técnicas de compilação, redes, etc.).

Durante o transcurso de 1969, o Departamento de Investigação Científica de Computadores dos Laboratórios Bell utilizou um computador 645 da General Electric com um sistema operacional denominado "MULTICS". Multics pode ser considerado como um dos principais sistemas interativos **multiusuário**, isto é, que permite a utilização simultânea do computador por vários usuários e que além disso, por ser interativo, oferece uma resposta quase imediata à petição de uma determinada ordem.

Anteriormente, o método de trabalho era substancialmente diferente, visto que os sistemas operacionais implementados funcionavam por lotes. O usuário datilografava as ordens destinadas ao computador em fichas perfuradas, que eram lidas pelo computador e algum tempo depois (que podiam ser vários minutos ou várias horas) obtinham-se os resultados em papel impresso. Da-

da a lentidão do processamento, o Multics estabeleceu um importante avanço; todavia, este sistema operacional necessitava de características importantes para os programadores.

Em 1969 Ken Thompson e Denis Ritchie criaram um sistema operacional com o propósito de dissimular alguns problemas de complexidade que se derivavam da utilização de grandes computadores. Este sistema foi chamado UNIX e estabelece a simplificação do diálogo entre o homem e a máquina.

Thompson e Ritchie consideraram que a maior parte do desenvolvimento de programas (mais de 50%) era empregado em criar arquivos fontes e compilá-los. Para tais tarefas não é necessária a utilização de uma grande máquina, bastando apenas um computador pequeno que disponha de grande memória em periféricos onde armazenar os arquivos.

Aspecto importante a considerar é que o programador disponha de um método simples para a criação, atualização e manuseio dos arquivos; esta é uma das características essenciais do UNIX: um sistema de arquivos com estrutura hierárquica em forma de árvore. Esta ferramenta permite a criação de um arquivo sem necessidade de definir parâmetros do mesmo (extensão, formato, etc.); pode-se localizar, além disso, de forma automática, qualquer arquivo em qualquer diretório do sistema.

Uma vez criado o arquivo fonte, é necessário compilá-lo e para tal tarefa tampouco se requer um grande computador, que unicamente será necessário na hora de executar grandes programas. Foram desenvolvidos, portanto, em Bells Labs uma série de programas de utilidade para facilitar as comunicações entre os grandes e pequenos computadores. Estas utilidades formam parte do sistema UNIX e são acessíveis pelo usuário. Estes programas permitem a tradução do código interno utilizado pelos pequenos PD P-11 da Digital, ao código EBCDIC usado pelos grandes computadores IBM.

Concluindo, com o UNIX se objetivou dotar as pequenas máquinas com um sistema interativo comparável ao dos grandes computadores da época.

Desenvolvimento do sistema

A primeira versão do UNIX foi realizada durante os anos 1967-1970 nos computadores PDP-7 e PDP-9 da Digital Equipment. Esta versão escrita em linguagem Assembler, continha os

principais conceitos de sistemas como CTTS, TENEX, MULTICS. No princípio, somente permitia a utilização do computador em monousuário e o desenvolvimento do software fundamental (Assemblers, compiladores, editores de texto e sistema) estabeleceu cerca de cinco anos-homem de programação.

Em 1971, foi desenvolvida a segunda versão do UNIX. Foram utilizados para isto o PDP-11/40 e o PDP-11/45. Nesta nova versão foi introduzida como característica mais importante, a exploração do UNIX em forma de multiprogramação.

Thompson desenvolveu uma linguagem baseada no "BCPL" com a finalidade de que fosse transportável e o denominou "Linguagem B". No mesmo ano de 1971, Ritchie a modificou e o novo resultado foi chamado "Linguagem C". A linguagem C foi utilizada tanto para a escrita de sistemas e de software básico como para aplicações do usuário.

A reescrita na linguagem C do Sistema UNIX foi ralizada em 1973 para algumas máquinas da gama PDP-11, estabelecendo-se assim, novas funções para o sistema. Em 1974, foi escrita uma versão simplificada do UNIX para o microcomputador LSI-11/03.

Como mencionamos anteriormente, o software inclui muitos programas desenvolvidos na linguagem C. Alguns foram incorporados como novas ordens do sistema para usuário ou técnicos de sistemas, enquanto que outros executam funções específicas. Por exemplo, um programa atualiza um banco de dados em conjuntura de real time, enquanto que outro proporciona possibilidades de edição de textos. Os professores e estudantes universitários escreveram numerosos programas C compátiveis com o software do UNIX. A Western Electric vende licenças para o sistema UNIX com aproximadamente 250 programas.

Até o nascimento da versão VI do UNIX, seu uso era interno e restrito aos laboratórios Bell. Estava espalhado pelos diferentes departamentos, que lhe foram incorporandos modificações conforme suas respectivas necessidades (MERT, PWD/UNIX).

Em 1975, o sistema UNIX começou a ser divulgado. Sua comercialização limitou-se ao envio de uma fita magnética e um exemplar de cada manual. Os laboratórios Bell não garantiram, então, nem a instalação e nem a manutenção do sistema. O **PWD** ("Programmer's Workbech") desenvolveu-se ao mesmo tempo que a versão VII do UNIX para um equipamento diferente do utilizado por Tompson e Ritchie. Utilizou-se para centros de cálculo onde o desenvolvimento era feito com PWD/UNIX em microcomputadores PDP-11 e a execução se realiza em sistemas IBM 370 e UNIVAC 1100. As inovações que auxiliam o PWD com relação à ver-

são VI do UNIX são:

- modificações nas funções de organização de usuários em projetos de grande tamanho;
- criação de utilitários para gestão das fontes ("SCCS") e o envio de trabalhos a distância.

Nos laboratórios Bell o sistema PWD/UNIX foi posto à disposição do serviço B.I.S.P., que possui uma rede de PDP-11/45 e PDP-11/70 conectados com o IBM 370/168, XDS SIGMA5 e UNIVAC 1100 utilizados por grande quantidade de usuários.

A versão VII estabeleceu uma troca importante em relação à VI, introduzindo uma série de modificações que podemos resumir em:

- superação das limitações quanto ao tamanho dos arquivos;
- intenção de portatibilidade com o propósito de transportar o UNIX a diferentes máquinas:
- desenvolvimento de novos utilitários.

Até então UNIX somente havia sido instalado em máquinas PDP-11, contudo, chegou-se à conclusão de que era necessário transportá-lo a outras máquinas. O primeiro transporte foi realizada, em 1977, sobre um computador INTERDATA 8/32. O fato de que fosse necessário a transportabilidade determinou uma série de modificações:

- a nível da linguagem C, foram realizadas as mudanças oportunas para que os programas se tornassem independentes na medida do possível das características físicas da máquina;
- a nível do compilador foram introduzidas técnicas que facilitaram a adaptação do mesmo às diferentes máquinas;

- a nível do sistema, pretendeu-se isolar as partes mais dependentes da arquitetura da máquina do resto.

Devido ao aumento de potência e ao barateamento do custo dos minicomputadores, o sistema operacional UNIX tornou-se popular rapidamente. Estas máquinas foram utilizadas imediatamente para o controle de experiências de laboratório, suporte de desenho auxiliado por computador, supervisão de redes de telecomunicações e execução de funções comerciais. O desenvolvi-

mento de um software que cumprisse estas aplicações específicas determinou aos programadores um novo desafio e o sistema UNIX era oferecido como uma ferramenta efetiva para consegui-lo. Até 1978 estavam funcionando em universidades, departamentos governamentais e nos laboratórios Bell mais de 500 instalações do sistema UNIX.

Com as modificações descritas anteriormente, surgiu em 1978 a versão sétima do UNIX (UNIX/V7) para os computadores PDP-11 UNIX/V32 e VAX 11/780. O UNIX Sistema III, que apareceu em 1982, inclui além disso:

- utilitários do PWB/UNIX;
- uma distribuição para máquinas da gama da Digital Equipment (do PDP-11/23 ao VAX 11/780);
- mecanismos de comunicação entre processamentos.

O sistema V, que surgiu em 1983, oferece também a licença de uso, instalação, manutenção e atualizações do sistema. Atualmente, é importante destacar o sistema UNIX desenvolvido pela Universidade de Berkeley (versão 4.1. BSD), que gestiona memória virtual. Estes programas foram adotados pelos laboratórios Bell e são incluídos nas novas versões do UNIX. Uma destas importantes contribuições é o editor do texto **vi**.

Definitivamente, existem duas opções possíveis na escolha do UNIX como ferramenta de trabalho:

- os sistemas que trabalham com o UNIX transportado a uma máquina em particular. Neste caso, é necessário o pagamento de licença e de Royalties aos laboratórios Bell;
- os sistemas que possuem os mesmos serviços que o UNIX mas que foram suficientemente modificados para que sejam considerados alheios aos laboratórios Bell.

Os programas UNIX

Os programas do sistema UNIX estão classificados da seguinte forma:

- O núcleo, que planeja tarefas e gestiona o armazenamento de dados.

- A Shell é um programa que relaciona e interpreta as ordens tecladas por um usuário do sistema. Interpreta solicitações de usuários, chama programas da memória e os executa ao mesmo tempo ou em séries de canalização chamadas "tubulações" (pipes).
- Os programas de utilidade executam uma variedade de subrotinas e diversas funções especiais de manutenção do sistema.

O sistema UNIX pode ser completado ou modificado por todos aqueles que disponham de licença de acesso aos códigos fonte do mesmo. Um elevado número de programadores foram incorporando programas ao sistema, incluindo os da Universidade da Califórnia. Estes programas, e os que irão aparecendo, incrementarão a ampla biblioteca de software existente no UNIX.

CAPÍTULO II

O USUÁRIO FRENTE AO SISTEMA UNIX

Quando se pretende acessar ao sistema operacional UNIX a primeira coisa que devemos fazer é nos identificar ao mesmo. Isto é conseguido com um identificador que permita ao sistema comprovar se se possuem os direitos próprios de acesso, identificador que é atribuído pelo administrador do sistema. Também é necessário dispor de um terminal do tipo conversacional conectado ao sistema para poder efetuar essa tarefa de apresentação ao sistema.

Estes procedimentos são denominados **login**, e podem ser efetuados através do comando **login** < **nome do usuário** > ou então automaticamente depois do engate e conexão do terminal ao sistema. Explicaremos a utilização do teclado do terminal com seus caracteres especiais e simularemos uma sessão de trabalho UNIX para ir aprofundando e desenvolvendo alguns comandos básicos e fundamentais.

Terminais e sua utilização

A comunicação entre o sistema operacional UNIX e o usuário se realiza, fundamentalmente, através de um terminal ou console (entrada/saída). O sistema UNIX trabalha sob a forma de transmissão "**full duplex**", isto é, os caracteres que são digitados no console são enviados ao sistema e este responde reenviando-os à tela, os quais são visualizados pelo operador. Em casos concretos

e especiais, como pode ser a digitação de uma palavra secreta de passagem ou acesso ao sistema, este "eco" na tela é eliminado.

Muitos dos caracteres que podem ser digitados são diretamente visualizáveis na tela, mas existem caracteres especiais que possuem outra interpretação e significado para o sistema. Um dos mais importantes é o caracter gerado pela tecla de RETURN. Este caracter indica ao sistema o fim da linha de entrada, movendo o cursor ao princípio da linha seguinte. O caracter RETURN deve ser enviado ao sistema antes de que este interprete a seqüência de caracteres enviada previamente. Como podemos observar, é um dos caracteres de controle (caracteres invisíveis), que produzem uma série de efeitos especiais em nosso terminal.
Este caracter RETURN está implementado diretamente sobre o teclado da maioria dos terminais. Outros caracteres de controle devem ser gerados pressionando uma tecla especial denominada CTRL, CTL ou CTNL e outro caracter, que normalmente costuma ser uma letra. O caracter RETURN é equivalente a pressionar a tecla CTRL seguida de "m". Existe outra série de caracteres de controle que definiremos a seguir:

- < CTRL > d ⟶ Indica ao processamento que terminou a entrada de caracteres. Não há mais caracteres na entrada.
- < CTRL > h ⟶ Efeito de backspace. Utiliza-se para a correção de erros de teclagem.
- < CTRL > i ⟶ Tabulador. Avança o cursor da tela até o próximo tabulador definido (tab stop). No UNIX a distância entre os tabuladores é de 8 espaços.
- <DELETE> / <RUBOUT> / <CTRL >c ⟶ Em muitos sistemas UNIX este caracter produz a finalização de um processamento sem esperar o seu término
- <BREAK> ⟶ Esta tecla, dependendo de como está definido o terminal, produz efeitos similares ao DELETE ou RUBOUT.

Uma sessão de trabalho com UNIX

Vamos simular uma sessão de trabalho com o sistema e ver algumas de suas características. Na seqüência de cada comando

inclui-se uma explicação do mesmo (tenha em conta que essa parte não entra no diálogo estabelecido com o sistema):

```
                                                    Efetuamos a conexão ao sistema ligando o
                                                    terminal ou a comunicação telefônica. O sis-
Sistema UNIX - PDP-8                                tema lhe responderá com a mensagem:
login: provas                                       Nome do usuário e RETN.
Password:                                           Não há eco na tela.
You have mail.                                      Aviso de que há correspondência.
$                                                   Pode-se digitar comandos
$                                                   Se foi pressionado RETN.
$ date                                              É solicitado data e hora.
Sat May 17 12:20:54 EDT 1986
$ who                                               Quem usa o sistema.
desenvolvimento      tty00   May   17   12:04
provas               tty02   May   17   09:25
pedro                tty03   May   17   11:00
processamentos       tty05   May   17   10:30
$ mail                                              Solicita-se a correspondência.
From pedro Sat May 17 11:02 EDT 1986
Telefone-me amanhã, por favor.
Obrigado. Saudações.
?                                                   RETN seguinte mensagem
From director Sat May 17 08:33 EDT 1986
Deixe os informes em minha mesa.
Não vou estar o dia todo.
? d                                                 Elimina-se a mensagem
$                                                   Não há mais correspondência.
$ mail director                                     Envio da correspondência ao diretor
amanhã recolherei os novos discos.
ctrl-d                                              Fim da correspondência ao diretor
$ ctrl-d                    saída do sistema
login:
```

Nas proximas seções desenvolveremos esta sessão e incluiremos outros comandos de utilização dentro do sistema.

Entrada ao sistema (login)

Antes de continuar, vamos definir e explicar o conceito do usuário e grupo de usuários do sistema. O sistema associa a um usuário com seu nome identificador, isto é, não reconhece pessoas físicas mas, digamos, razões sociais. Portanto, pode ter-se diferentes terminais do sistema trabalhando para o mesmo usuário. Podemos definir o usuário como a entidade lógica do sistema sobre a qual se vai produzir todo o trabalho do computador em uma sessão com o mesmo. Qualquer arquivo, programa ou documento criado dentro de uma sessão pertencerá ao usuário no qual se encontra e todos os processamentos serão executados dentro desse usuário ao qual o operador foi conectado.

Um usuário pertence a um agrupamento de usuários, o **grupo**, de maneira que-se pode compartilhar com os usuários de um mesmo grupo os direitos de acesso a determinados arquivos comuns. Tudo isto tem inter-relação com as permissões de acesso aos arquivos que serão explicados no capítulo correspondente. Assim pois, um usuário pode ser proprietário de um arquivo, membro do mesmo grupo que o proprietário do arquivo ou inclusive não ter nada a ver com ele (restante dos usuários). A atribuição de um usuário a um grupo se produz no momento da criação do mesmo pelo administrador do sistema e dentro do arquivo **/etc/passwd**. Cada usuário possui um número identificador do mesmo, que é atribuído pelo sistema no momento da criação do usuário. O número do usuário "0" corresponde ao administrador do sistema e seu nome é **root**. Trabalhando neste usuário se possuem os privilégios máximos do sistema e podem ser efetuadas tarefas de manutenção do mesmo. Seu nome comum de identificação é **super user**. Este superusuário se encarregará de manter a integridade do sistema de arquivos, restaurá-lo em caso de queda do mesmo ou erro de operação, manter os usuários que irão ser definidos e, em geral, efetuar tarefas de manutenção do sistema. Existem comandos que somente podem ser efetuados pelo superusuário.

Trabalhando dentro de qualquer usuário que não seja **root** pode-se adquirir os privilégios deste através do comando **su**.

```
$ date
Sat May 17 14:56:10 EDT 1986
$ date 1810                          ordem de mudar a hora.
date: no permission                  não há previlégio do sistema
Sat May 17 14:56:20 EDT 1986
$ su                                 entrada no privilégio máximo
Password                     solicita a senha do
                             usuário root.
#  <RETN>                            Prompt # do super usuário
# date 1920                          Solicitação de mudança da hora
Sat May 17 19:20:00 EDT 1986         Hora mudada no sistema
# ctrl-d                     Volta ao usuário
                             normal
$                                    Prompt do usuário normal
```

Como vimos, a palavra chave do usuário **root** deve ser conhecida por poucas pessoas, visto que nos garantirá a integridade do sistema ante possíveis operadores operadores ineficientes ou malintencionados que poderiam entrar no superusuário. Quando se tem o prívilégio do superusuário, o sistema não efetua controle algum sobre os processamentos que lhe são ordenados executar,

com o que a responsabilidade do mesmo é grande.

O sistema operacional possui uma lista definida de usuários (à parte dos de gestão e base do sistema), definidos por um nome, uma palavra chave de acesso e uma série de atributos internos. Para a entrada no sistema é necessário o conhecimento do nome e da palavra de acesso ao usuário.

Suponhamos que o terminal utilizado tenha a possibilidade de trabalhar com caracteres alfabéticos em minúscula e maiúsculas. Devemos comprovar que todos os switches do terminal estão corretamente instalados para trabalhar sob o sistema operacional UNIX (consultando o manual do terminal pode-se obter a configuração correta). Entre estes, os mais importantes são a velocidade de transmissão (bauds), modo "full duplex", paridade e maiúsculas-minúsculas. Uma vez conectado o terminal, aparecerá no mesmo a mensagem:

login:

Em resposta a esta mensagem, o operador deve digitar o nome do usuário onde se vai produzir a sessão de trabalho e pressionar a tecla RETURN. Se o usuário (que será definido pelo administrador do sistema) possui uma palavra de acesso, o sistema a solicitará e o operador deverá digitá-la corretamente, e o sistema não produzirá eco na tela para preservar o segredo de tal palavra.

A culminância do processamento de entrada será a apresentação na tela de um "prompt"(caracter significativo de que se está sob o controle do sistema operacional) que normalmente é o caracter "**$**" ou "**%**", mas que pode ser trocado ou redefinido pelo administrador do sistema mediante o comando apropriado. Este prompt é enviado por um programa denominado "**intérprete de comandos (command interpreter) ou Shell**" que é a interface do usuário com o sistema operacional. Antes da visualização do prompt pode-se obter a data, uma mensagem de aviso de que temos correspondência pendente ou então indicação do tipo de terminal que se está utilizando.

A digitação no teclado do nome do usuário deve ser efetuada em minúsculas, visto que se for feita em maiúsculas, o sistema utilizará, posteriormente, esse mesmo tipo de letra em todas as comunicações do usuário, entendendo que o terminal não permite a possibilidade de trabalhar em maiúsculas-minúsculas.

Existe um comando para a mudança da senha ou acesso a um usuário. Para isto, devemos entrar (login) no usuário desejado e digitar o comando **passwd:**

```
$ passwd
Old password:
New password:
Retype new password:
$
```

Uma vez digitado, o sistema solicitará a senha atual do usuário, que deverá ser digitada pelo operador e não produzirá eco na tela. Em caso de erro na mesma, o sistema eliminará o comando, em outro caso pedirá nova senha, que tampouco fará eco na tela. Uma vez digitada voltará a solicitá-la, de tal forma que devemos digitar a mesma senha anteriormente digitada.

Isto se deve a motivos de seguraça do sistema. Se coincidirem, a nova senha será tomada como a atual para o acesso ao sistema. Rememore esta palavra de acesso, dado que, se esquecê-la, deverá solicitar sua troca ao administrador do sistema ou à pessoa encarregada de sua manutenção.

Introdução de comandos

Como dissemos, anteriormente, cada vez que digitamos no teclado os caracteres que compõem um comando, o sistema os interpretará e executará uma vez que tenhamos pressionado a tecla RETN (RETURN), produzindo-se a interpretação do comando enviado.

Uma vez recebido o prompt do sistema, podemos começar a introduzir comandos, que nada mais são do que requerimentos ao sistema para que efetue um determinado processamento. Como pudemos ver na primeira sessão de trabalho representada, o operador digitou o comando **date** e pressionou a tecla RETN, o que produziu uma saída na tela da data e hora do sistema.

O comando seguinte foi **who**, que é uma solicitação ao sistema para que mostre a lista de usuários que atualmente estão ativos no mesmo. A primeira coluna da saída corresponde ao nome do usuário que está ativo. A segunda é o nome que o sistema atribui à conexão utilizada pelo usuário (tty é um sinônimo de "teletype" ou "terminal"). O restante são os dados da data e hora da conexão do usuário. Poderíamos também ter usado outra série de comandos:

```
$ who am i                    Dados do próprio usuário
provas tty02 May 17 09:25
$ whom                        Não existe esse comando
whom: not found               O sistema nos avisa da inexistência.
```

Como podemos observar, o sistema nos avisa no caso em que tenhamos solicitado a execução de um comando que não existe.

Pode ocorrer que nosso terminal atue de forma estranha, aparecendo em dobro os caracteres na tela ou não posicionando o cursor no princípio da linha. Pode-se tentar corrigir estes defeitos desligando e religando nosso terminal. Também podemos utilizar o comando **stty** para efetuar o tratamento inteligente dos caracteres de tabulação. A descrição do funcionamento deste comando escapa aos propósitos deste livro, mas em caso de necessidade pode-se utilizar os Manuais originais do UNIX. Também podemos utilizar o comando **tabs** para obter idênticos resultados.

Controle de erros de escrita

No caso de detectar um equívoco ao teclar um caracter antes de pressionar RETURN, há duas possibilidades de correção: eliminar os caracteres um a um ou eliminar a totalidade do comando digitado. Para este propósito, existem dois caracteres que produzem um efeito especial para esta correção:

caracter # → eliminação de caracter (erase character)
caracter @ → eliminação da linha digitada (kill character)

Se o operador introduz o caracter de eliminação de linha ao final de um comando, este será ignorado completamente e o digitado começará a ser interpretado em continuação do caracter "kill".

```
$ ddate @                                   Ignora-se o comando "ddate "
date                                        continua em nova linha.
Sat May 17      16:45:50 EDT 1986
$
```

No caso em que o operador digite a eliminação do caracter depois de qualquer outro caracter, este será ignorado na hora de interpretar o comando, com o qual podemos fazer correções no comando antes de seu envio através da tecla RETN:

```
$ dd# atte# #e
Sat May 17      16:45:50 EDT 1986
$
```

Os caracteres de eliminação de linha e caracter são muito dependentes do sistema e terminal que está sendo utilizado. Em muitos sistemas, a eliminação de caracter foi assimilado à tecla de

"backspace", que trabalha muito melhor nos terminais com tela de vídeo.

No caso em que desejamos incluir estes caracteres especiais de eliminação dentro de um texto, por exemplo, deverão ser precedidos pelo caracter "backslash" " \ " (caracter de escape), de tal forma que deveremos digitar " \ @" ou " \ # ". Este caracter de escape (\ ) indica ao sistema que o caracter que vem a seguir deve ser tratado de uma forma especial.

A introdução de comandos pode ser efetuada toda vez que o processamento em execução produz saída à tela. Devido a isto, podemos produzir a mistura na mesma de caracteres correspondentes à entrada do operador e saída do processamento. Esta situação não tem nenhum problema, salvo a desordem da saída de caracteres da tela. Isto é devido à existência de um "buffer" de armazenamento dos caracteres de comandos que o operador está digitando e a possibilidade de multiprocessamento que implementa o sistema operacional UNIX.

Existe um comando de nome **stty** que nos permite, por um lado, visualizar os critérios de comunicação com o terminal e, por outro, a modificação destes critérios.

```
$ stty
speed 9600 baud
interrupt = 'DEL' quit = '^\' erase = '^H' kill = '^U'
even odd -n echo -tabs ffl
$ stty ek
$ stty
speed 9600 baud
interrup = 'DEL' quit = '^\' erase = '#' kill = @
even odd -n echo -tabs ff1
$
```

Ao executar o comando **stty** nos é mostrada a velocidade de transmissão com o terminal, os caracteres definidos pelo sistema para a detenção de um processamento, eliminação de caracter e linha, saída do sistema, paridade, estado do eco e tubuladores definidos.

A execução do comando **stty ek** (ek = erase kill) restaura os caracteres de eliminação de caracter e linha ao estado normal do sistema. Se voltarmos a efetuar **stty** nos mostra o novo estado do terminal.

Parada e pausas de um processamento ou programa

Através da tecla DELETE ou RUBOUT podemos produzir a parada de um processamento que foi lançado pelo usuário atra-

vés do envio de um comando determinado. Em alguns processamentos especiais, como o editor **ed**, esta tecla produz a parada da tarefa que está sendo efetuada dentro do processamento, porém nos mantém dentro dele. Desligando o terminal também podemos produzir a parada de numerosos processamentos.

Se quisermos efetuar a parada momentânea de um processamento podemos teclar **CTRL-s** (lembre que se deve pressionar simultaneamente a tecla do CTRL e o caracter "s"). Com isto podemos conseguir a parada da visualização na tela de alguns dados que pelo andamento do processamento irão desaparecer da mesma. Para continuar o processamento, normalmente, deveremos digitar **CTRL-q**.

Saída do sistema

A forma mais normal de acabar uma sessão de trabalho com UNIX e sair do usuário no qual estávamos é digitar **CTRL-d**, ao invés de utilizar um comando. Esta seqüência indica à Shell, que não irão mais ocorrer entradas e que se deve deixar o sistema livre para a conexão de outro possível usuário.

Correio entre os usuários

O sistema operacional UNIX dispõe de uma forma de correio-comunicação entre usuários, amparada pelo comando **mail**.

No caso em que o usuário onde entramos tenha correio pendente, o sistema o avisa através de uma mensagem, sendo optativo ao operador a visualização ou não do mesmo. Obtém-se a correspondência executando o comando **mail**.

Depois de cada mensagem do correio mostrado pelo comando, este espera a ação que o mesmo irá desenvolver. Há duas respostas básicas: uma a eliminação do correio visualizado, através da digitação da seqüência **d** <**RETN**> , e outra a visualização da seguinte mensagem pendente, que é obtida digitando **RETN**. Outras possibilidades do comando **mail** são:

? p	Volta a visualizar a mensagem anterior.
? s <nome arquivo>	Armazena a mensagem no arquivo cujo nome lhe é dado.
? q	Sai do comando **mail** e retorna à Shell do UNIX.

No caso de desejármosenviar correio à outros usuários. deveremos digitar o comando **mail** seguido do nome do usuário o qual se deseja enviar. O cursor se posicionará no princípio da linha seguinte e estará pronto para a digitação da mensagem. O final de uma linha se marca com RETN e a mensagem continua na linha seguinte, Para finalizar o texto da mensagem, digitaremos RETN e CTRL-d, retornando à Shell.

O comando **mail** também permite o envio do correio a múltiplos usuários, à usuários de outras máquinas ou o envio de cartas já preparadas.

Comunicação imediata com outros usuários

Durante o trabalho normal de uma sessão, podemos produzir em nossa tela mensagem do tipo: **Message from pedro tty06**, acompanhada por um silvo do terminal. Isto nos indica que o usuário "pedro" deseja estabelecer uma comunicação conosco. Para especificar ao sistema que estamos de acordo, o usuário deverá teclar: $ write pedro.

Este comando estabelecerá uma dupla via de comunicação entre ambos os usuários, de tal forma que todas as linhas que são digitadas dos mesmos aparecerão no terminal do outro usuário.

Como o comando **write** não estabelece nenhuma regra de comunicação, os usuários devem ser regidos por algum tipo de protocolo. No caso que vamos mostrar a seguir, a seqüência **(f)** indica fim do envio do texto (o outro usuário pode enviar mensagens) e a **(ff)** que se está pronto para o fim da comunicação:

Terminal do pedro	**Terminal de provas**
$ write provas	
	$ Message from pedro tty06
	$ write pedro
Message from provas tty02	
Lembra-se da entrevista de hoje? (f)	
	Lembra-se da entrevista de hoje? (f)
	Às dez @
	Às onze. Correto? (f)
Às onze. Correto? (f)	
Correto (ff)	
	Correto (ff)
	ctrl-d
EOF	
ctrl-d	
	$ EOF
$	

Através da tecla DELETE se pode sair do comando **write**. Note que os erros de digitação corrigidos pelo usuário "provas" não

aparecem no terminal de "pedro", mas somente é mostrada a linha corrigida.

No caso em que se tente a conexão com um usuário que não está conectado ao sistema (login) ou que estando conectado não deseja estabelecer a conversação, o sistema avisará isto.

Muitos dos sistemas UNIX possuem um comando especial de nome **news**, para visualizar as possíveis notícias relevantes ao sistema, seu desenvolvimento e possíveis melhoras e incidências.

Manual original do sistema UNIX

O manual do usuário "**UNIX Programmer's Manual**" descreve detalhadamente o conjunto de funções e comandos necessários para o completo manejo do sistema. A Seção 1 do Manual inclui a descrição de comandos UNIX. A Seção 2 trata dos "system calls" do sistema e a Seção 6 inclui os jogos implementados no UNIX. As demais seções ocupam-se de funções que podem ser tratadas por programadores em linguagem C, formatos de arquivos e manutenção do sistema (tarefa do administrador do mesmo).

Freqüentemente, o Manual original do UNIX é armazenado no próprio sistema e pode ser lido sob o comando **man** seguido pelo nome do comando cuja sintaxe e descrição se deseja:

```
$ man who
.....                  Visualização da explicação detalhada do
.....                  comando who, servindo de ajuda ao usuário.
$ man man              O mesmo do próprio comando man.
.....
.....
$
```

O sistema, assim mesmo, possui um comando que facilita o aprendizado do sistema operacional, cujo nome é **learn**. Este comando inclui o estudo do sistema de arquivos, comandos básicos, editor **ed** do sistema, preparação de cartas e documentos e, inclusive, linguagem C.

CAPÍTULO III

OS EDITORES ED E VI

O editor "ed" é o mais simples dos editores incorporados pelo sistema operacional UNIX. Trata-se de um "editor de linhas" no sentido de que seus comandos foram concebidos para trabalhar sobre uma única linha de texto, diferente dos "editores de tela", preparados para efetuar deslocamentos, buscas e inserções em qualquer ponto da tela do computador e, portanto, do texto.

Para editar um arquivo denominado "nome" temos que teclar:

```
ed nome
```

com o qual o sistema devolverá a resposta

```
? nome
```

Se se tratasse de um arquivo de nova criação, ou

```
nome
1420
```

se "nome" fosse um arquivo já existente e tivesse um tamanho de 1420 caracteres.

Para gravar um arquivo simples, como

```
main ( )
{
        printf ("hello, world\n");
}
```

Procederíamos da seguinte forma:

```
$ ed nome.c
? nome.c
a
main ( )

        printf("hello, world\n");


w
q
$
```

O sinal "$" corresponde ao prompt da Shell, e indica que está à espera de receber um novo comando. A linha "ed nome.c" chama o editor "ed" para modificar o arquivo "nome.c", respondendo "ed" com a linha "? nome.c" para indicar que se trata de um novo arquivo.

A linha que contém um "a" corresponde ao comando "append" (acrescentar) do editor, indicando que se vai acrescentar texto ao conteúdo do arquivo "nome.c". Em seguida, introduz-se o texto até a aparição de uma linha com um ponto "." que indica ao editor que finaliza a introdução de texto e passa-se do "modo texto" ao "modo comando", para seguir introduzindo comandos do editor.

Finalmente, utilizam-se os comandos "w" ("write", escrever) para gravar em disco o conteúdo do arquivo, e "q" ("quit", sair) para finalizar a sessão de edição.

Se agora reeditarmos o arquivo "nome.c", o sistema nos responderá com a linha

```
$ ed nome.c
34
```

Indicando que "nome.c" contém 34 caracteres. Se teclarmos o comando do editor "1,$p" será mostrado na tela o conteúdo da primeira (1) à última ($) linha do arquivo.

Para modificar a terceira linha e trocar "hello" por "alô", procederíamos pressionando "3s/hello/alo/p", com o qual o editor nos responderia com a linha:

```
printf("alo, world\n");
```

O editor "ed" é utilizado no UNIX para manter e modificar arquivos especiais, que contenham caracteres de controle (caracteres ASCII cujo código é inferior ao do caracter espaço em branco, ou

ASCII 32 em decimal), e para modificar arquivos em modo não interativo.

A edição do mesmo arquivo "nome.c" através de editor "vi" seria realizado como:

```
$ vi nome.c
"nome.c" [new file]
```

Eliminando-se a tela, e apresentando o caracter " " da primeira à última linha, na qual apareceria o nome do arquivo e o texto "new file", indicando que se trata de um novo arquivo.

Seguidamente, seria introduzido o comando "a" ("append", acrescentar)e começaria a introdução do texto. Para finalizara mesma, pressiona-se a tecla ESCAPE "ESC"e passa-se ao "modo comando", teclando posteriormente "wq" para gravar o arquivo e finalizar a edição.

Para o movimento de uma linha à outra, podemos utilizar as flechas dispostas sobre o teclado, em lugar de referirmo-nos a uma linha em particular como no caso do editor "ed".

"vi" é tão-somente um programa editore não resulta, portanto, tão completo como pode ser um tratamento de textos como o "Wordstar" do CP/M. Permite o manuseio de instruções da Shell ou o emprego de comandos do "ed" para aumentar sua flexibilidade e repertório de comandos.

Por questões de espaço, não nos aprofundaremos no manejo doseditores"ed" e"vi",remetendo ao leitor interessado ao apêndice, onde se inclui um amplo resumo dos comandos do UNIX, figurando ali as opções mais freqüentes destes editores.

CAPÍTULO IV

ARQUIVOS DO SISTEMA UNIX

Tudo no Sistema UNIX são arquivos. É a idéia mais simples e a mais clara que se pode ter no princípio sobre o referido sistema operacional. As maiores discussões no início da criação do sistema UNIX se concentraram na estrutura interna que mantivesse o sistema organizacional de arquivos, de tal forma que fosse potente e simples a sua utilização. Foi escolhida como base desta filosofia a implementação de um sistema baseado em um **pequeno conjunto de boas idéias,** que desse como resultado um poderoso sistema de arquivos.

Neste capítulo foram cobertos alguns detalhes sobre a estrutura, criação e organização de arquivos do sistema, diretórios, permissões de acesso. Será incluída a descrição de numerosos comandos do sistema que efetuarão tarefas de manutenção e manipulação de arquivos ou que, definitivamente, mantenham uma forte ligação com o sistema de arquivos UNIX.

Noções básicas sobre arquivos

A definição mais clara e simples de um arquivo é a de uma seqüência de bits, entendendo por bit a unidade menor de informação processada por qualquer equipamento que trabalhe sob o sistema operacional UNIX. No princípio, a estrutura da informação que está armazenada no arquivo somente é relevante para o programa.

Para o sistema operacional UNIX, uma série de caracteres teclados no console, qualquer das impressoras do sistema, os conjuntos de informação (arquivos de dados) armazenados no disco rígido, uma fita magnética (streamer) ou os dados que fluem através dos **pipe** do sistema são considerados como arquivos, tendo um tratamento de acordo com a estrutura e utilização de cada um.

A melhor forma de conhecer e aprofundar no sistema de arquivos UNIX é criar um e efetuar nele uma série de processamentos. Façamo-lo através do comando do sistema **ed** (entrada ao editor "ed").

```
$ ed
a
ABCDEFGHIJKLMNO
0123456789

w texto
27
q
$ ls -l texto
-rw-r--r-- 1 prova          27 May 12 09:54 texto
$
```

O arquivo "texto" contém 27 bytes, que são os caracteres teclados pelo usuário que o criou. Para visualizar o conteúdo do arquivo utilizaremos o comando **cat:**

```
$ cat texto
ABCDEFGHIJKLMNO
0123456789
$
```

Vamos visualizar o conteúdo interno (em bytes) do arquivo "texto". Através do comando **od** (octal dump) obteremos uma representação visível de todos os bytes que compõem o arquivo:

```
$ od -c texto
0000000 A B C D E F G H I J K L M N O   /n
0000020 0 1 2 3 4 5 6 7 8 9   /n
0000030
      $
```

Através da opção —**c** os bytes são interpretados como se fossem "caracteres". A opção —**x** nos mostrará os bytes em números hexadecimais (base 16):

```
$ od -cx texto
0000000 A B C D E F G H I J K L M N O   /n
          41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E OA
0000010 0 1 2 3 4 5 6 7 8 9   /n
          30 31 32 33 34 35 36 37 38 39 0A
0000020
```

Os dígitos numéricos que aparecem na parte esquerda da saída são as posições que ocupam os bytes dentro do arquivo, em notação hexadecimal. Determinamos que cada linha de saída representasse o conteúdo de 16 bytes do arquivo.

Ao final de cada uma das linhas do arquivo "texto", aparece um caracter "\n" que representa o "final da linha" **(newline)** que o usuário definiu através da tecla <RETURN>. Por convenção baseada no desenvolvimento da Linguagem C, este caracter é representado através da notação "\n". Internamente, armazena-se o caracter "12" em octal ou o "OA" no hexadecimal. O exemplo deste caracter especial é um dos muitos existentes:

• escape	(escapamento)		
• backspace	(espaço atrás)	\b	Octal 10 Hex. 08
• tab	(tabulador)	\t	Octal 11 Hex. 09
• carriage return	(retorno do carro)	\r	Octal 15 Hex. OD

É muito importante que saibamos distinguir entre o conteúdo interno de um arquivo, ou seja, os bytes armazenados no mesmo, e a interpretação deles é feita dependendo das situações ou os processamentos que tratem referido conteúdo.

Suponhamos que em um momento de nosso trabalho com o sistema, teclamos o caracter **backspace** e que tal caracter é definido como **eliminação de caracter**; o kernel interpreta que se pretende eliminar o caracter anteriormente teclado, com o qual desaparecerão da tela ambos os caracteres. Se, ao contrário, o usuário tecla a seqüência \<**backspace**>, o kernel interpreta que se deseja armazenar dentro de nosso arquivo de entrada (neste caso o console) o literal backspace e o byte representado por 08 em hexadecimal será armazenado, fazendo eco do mesmo no console (saída).

Se utilizarmos o comando **print** sobre um arquivo que contenha um caracter especial **backspace**, com saída no console, produzirá o movimento do cursor um espaço à esquerda. Se, ao contrário, sobre o mesmo arquivo se utiliza o comando **od** com a mesma saída, nos aparecerá como um byte de representação 08 em hexadecimal ou 10 em octal.

O sistema operacional UNIX não possui registros, nem contadores de bytes em registro, nem qualquer outro byte que não sejam os armazenados por nosso próprio programa de manutenção do arquivo. Esta é uma das regras básicas que foram estabelecidas durante o desenvolvimento do sistema de arquivos que suportaria o UNIX.

Por outro lado, vimos que ao final do arquivo não aparece nenhum caracter especial indicativo de **fim do arquivo**, mas que simplesmente tem-se a representação do mesmo. O sistema indica que terminou o arquivo simplesmente porque não há mais dados nele. O kernel mantém tabelas indicativas do comprimento total do arquivo, de tal forma que os processamentos encontrarão o final do arquivo quando tenham processado a totalidade de bytes do mesmo.

Localização dos arquivos

A maioria dos comandos que vamos explicar nas seções seguintes do capítulo, assim como em muitos outros, incluem a possibilidade de especificar alguns nomes de arquivos que lhe são passados como argumentos ao comando. Estes nomes de arquivos podem estar implícitos, ou seja, com seu nome completo, ou definidos com uma série de caracteres especiais, que vão obrigar a efetuar uma busca de arquivos, baseando-se em algumas destas regras que estão especificadas no núcleo do sistema. Vejamos uma tabela dos caracteres especiais substitutivos dentro do nome de um arquivo, junto com uma série de exemplos esclarecedores:

TABELA DE CARACTERES DE MÁSCARA UNIX (WILD CARDS)	
?	Em qualquer posição indica que é válido qualquer caracter, porém somente um.
*	Indica qualquer cadeia de caracteres à partir dessa posição, inclusive a cadeia nula.
[abc]	Na posição dada poderão existir os caracteres "a", "b" ou "c" e, em geral, qualquer dos caracteres situados dentro dos colchetes.
[^ abc]	Igual ao anterior, porém são válidos todos os caracteres exceto os contidos dentro dos colchetes.
[a-b]	Nessa posição é válido qualquer caracter que está alfabeticamente entre o caracter "a" e o caracter "b".
^	Faz com que os caracteres da máscara estejam no princípio do nome do arquivo.
$	Faz com que os caracteres da máscara estejam no final do nome do arquivo.

Vejamos alguns exemplos de utilização destas máscaras:

nom.?	⟶	válidos "nom.<caracter>" Exemplo: "nom.a" "nom.1" "nom.p" "nom.x". Não são válidos "nome.12" "no.1" "nom.ab"
pro.*	⟶	válidos "pro.<cadeia>" Exemplo "pro.c" pro.program" "pro.12". Não são válidos "nom.1" "program" "pro12" "pr.c"
arq.[123]	⟶	válidos unicamente "arq.1" "arq.2" "arq.3"
arq.[^ 123]	⟶	válidos todos os "arq.≤caracter>", exceto os "arq.1" "arq.2" e "arq.3"
uno a—d	⟶	válidos "unoa" "unob" "unoc" e "unod"

Vamos criar dois arquivos (sempre dentro do usuário de trabalho "prova") através do comando ed:

```
$ ed
a
Texto do documento uno
.
w texto
23
q
$ ed
a
O sistema UNIX em estudo
.
w arq.1
25
q
$
```

Relembremos que o contador de caracteres mostrado pelo comando inclui o caracter de fim de linha **(newline)**, que é a forma que tem o sistema de representar a digitação da tecla RETURN.

O comando ls nos mostra a lista dos nomes dos arquivos que o usuário mantém ou o sistema, não o conteúdo dos mesmos.

```
$ ls
arq.1
texto
$
```

```
$ ls
arq.1      texto
$
```

Coincide com os dois arquivos criados pelo usuário. São válidos os dois sistemas de representação que aparecem; dependendo da versão do sistema UNIX com a qual se trabalhe, usaremos uma ou outra. Os nomes estão automaticamente ordenados alfabeticamente. Como muitos dos comandos UNIX, **ls** possui opções que podem ser utilizadas para trocar o modo de ação do mesmo. As opções devem ir em seguida ao nome do comando e, normalmente, vão precedidas do caracter "—".

A opção **—t** produz uma ordenação dos nomes dos arquivos a partir da última modificação dos mesmos. As mudanças mais recentes, são colocadas no princípio.

A opção **— l** produz uma saída com informação muito mais completa sobre os arquivos.

```
$ ls -l
total 2
-rw-r--r-- 1 prova      25 May 17 18:21 arq.1
-rw-r--r-- 1 prova      23 May 17 18:20 texto
```

A primeira mensagem de saída ("total 2") nos indica o número total de blocos do disco utilizados para armazenar a informação dos arquivos que foram processados pelo comando. Um bloco é constituído por 512 ou 1024 caracteres. A cadeia "-rw-r--r--" nos dá informação das permissões de acesso aos arquivos. Neste caso, o proprietário do arquivo ("prova") tem permissão de leitura (r) e escrita (w), porém o restante de usuários somente tem permissão de leitura. A informação seguinte que observamos na saída ("1") é o número de enlaces **(links)** do arquivo. A cadeia "prova" nos indica o nome do usuário ao qual pertence o arquivo, isto é, o usuário que o criou. A cadeia seguinte de saída ("25" e "23") corresponde ao número de caracteres que compõem cada arquivo (bytes) e que vimos que coincide com o obtido pelo comando **ed** na criação de ambos os arquivos. A seguir mostra a data e hora da última modificação do arquivo e, por último, o nome do mesmo.

As opções do comando **ls** (assim como as de muitos outros) podem ser agrupadas, de tal forma que o comando **ls —lt** produz uma saída com informação completa dos arquivos e ordenados por data-hora da última modificação.

A opção **—u** nos dá informação do momento da última utilização do arquivo (não troca), **—r** produz a inversão da ordem de saída dos arquivos processados pelo comando.

Depois das opções do comando **ls**, podemos incluir um ou vários nomes de arquivo, de tal forma que a informação visualizada pelo comando pertencerá à lista de arquivos dada:

```
$ ls —l texto
—rw———r———r——— 1 prova          23 May 17 18:20 texto
$ ls —l arq.1
—rw—r———r——— 1 prova             25 May 17 18:21 fich.1
$ ls —l tex
tex not found
$
```

Como pudemos observar, no caso de não existir o arquivo solicitado, o comando nos mostra uma mensagem de aviso. Vejamos a seguir uma tabela das opções mais importantes do comando **ls.** Leve em conta que nesta parte do capítulo aparece a noção de diretório, que será mais tarde explicada; fica aqui seu resumo a nível de sua utilização no comando **ls:**

ls [—ltasdru] <nomes de arquivo>

—l lista em formato completo (longo) de informação;
—t lista por ordem de data-hora da última modificação. Primeiro as mais recentes;
—a lista todos os arquivos incluindo diretórios;
—s visualiza o tamanho dos arquivos em blocos. 1 bloco 512 caracteres (bytes);
—d informa sobre a situação dos diretórios;
—r lista em ordem inversa ao normal
—u lista por ordem de data-hora da última utilização dos arquivos. Primeiro a mais recente.

Alguns exemplos seriam:

```
$ ls —a
. .. .profile arq.1 texto
$ ls —s
total 2 1 arq.1   1 texto
$ ls —r
texto  arq. 1
$ ls —u
arq.1  texto
```

Vimos na opção **—a** que nos aparecem três arquivos "estranhos". O representado por ".", nos indica o diretório atual no qual nos encontramos. O ".." corresponde ao diretório pai do atual, isto é, o diretório do qual provém o "cacho". E o arquivo "**.profile**" é um dos arquivos "invisíveis" do usuário; contém uma série de parâmetros importantes para o controle do usuário.

O comando **ls** não costuma responder com mensagens de erro, inclusive no caso de uma digitação incorreta das opções. Quan-

do não são passados argumentos dos nomes dos arquivos, lista todos os arquivos do diretório atual. No caso de digitar a opção —l do comando esta imprime, entre outras informações, uma lista de 10 caracteres, dos quais o primeiro indica o tipo de arquivo que foi listado e os restantes 9 as permissões de acesso ao mesmo, que serão explicados posteriormente. Os tipos possíveis de arquivos, indicados pelo primeiro caracter da cadeia, são os seguintes:

- d a entrada é um diretório;
- b a entrada é um arquivo especial do tipo bloco;
- c a entrada é um arquivo especial do tipo caracter;
- — a entrada é um arquivo do tipo ordinário. Arquivo normal de usuário.

Relação do conteúdo de um arquivo

Existem numerosas formas para observar ou visualizar o conteúdo de um arquivo. Uma possibilidade é usar o editor do sistema **ed:**

```
$ ed texto 23
23                          Indica o número de caracteres
l, $p                       Listar linhas desde a 1 até a última
Texto do documento uno
q                           Saída do editor
$
```

Chamamos o comando **ed** passando-lhe o nome do arquivo que queremos editar ("texto"); nos indica o número de caracteres (bytes) que o compõem.

Existem numerosos casos nos quais não é viável o uso de um editor para a visualização da informação armazenada em um arquivo, por exemplo, no caso de que o tamanho do mesmo seja superior ao máximo processado pelo editor ou que se deseje a visualização de mais de um arquivo. Para estes casos existem várias alternativas que veremos a seguir.

O comando **cat** é o mais simples de todos os possíveis para visualizar informação de arquivos. Este comando lista o conteúdo de todos os arquivos que lhe foram passados como argumentos:

```
$ cat texto
texto do documento uno
$ cat arq. 1
O sistema UNIX em estudo
$ cat texto arq.1
Texto do documento uno
O sistema UNIX em estudo
$
```

O nome do comando procede da palavra concatenação. A concatenação é produzida dos arquivos passados como argumentos, sobre a saída (por default tela). A saída do comando ocorrerá sem nenhum tipo de pausa, todavia se o conteúdo do arquivo é grande, deveremos pressionar **ctrl-s** para parar a saída e manter a visualização que nos interessa.

O comando **cat** somente serve para a visualização do conteúdo de arquivos que estejam compostos por caracteres ASCII ou então caracteres que sejam imprimíveis. Se, por exemplo, um arquivo é listado com caracteres de controle, o efeito que podem causar em nosso terminal pode ser bastante estranho. O comando **cat** pode visualizar uma mensagem de erro, indicando que não encontra o arquivo solicitado:

```
$ cat salvador
cat: can't open salvador
$
ou então dependendo da versão:
$ cat salvador
salvador: cannot open
$
```

Em todos os sistemas UNIX existe um comando que permite acomodar a saída às características do terminal, de tal forma que nos permita a visualização do conteúdo por páginas. O nome do comando depende do sistema e pode ser **page, pg ou more.** Cabe ao usuário investigar a existência destes comandos em seu sistema.

Existe outro comando, de nome **pr,** que lista o conteúdo de um ou vários arquivos, acomodando-se às características das impressoras do sistema. Isto é, lista páginas de 66 linhas (11 polegadas) com a data e a hora da última modificação do arquivo, numera as páginas e coloca o nome do arquivo no princípio de cada uma. Entre cada um dos arquivos a serem listados, efetua um salto de página, de tal forma que cada arquivo começa no início de uma página.

```
$ pr texto arq.1
May 17 18:20 1986 texto Page 1
Texto do documento uno
        (62 linhas em branco)
May 17 18:21 1986 arq.1 Page 1
O sistema UNIX em estudo
        (62 linhas em branco)
$
```

Através deste comando, podemos produzir uma saída em multicolunas ou em paralelo. Por exemplo, **pr-3 arq. 1 arq. 2 arquivo**

efetua uma saída em formato de três colunas. A opção **-m** atribui a saída em colunas paralelas. Para mais informação sobre o comando deve-se utilizar o Manual original do UNIX (UNIX Programers Manual).

Devemos notar que este comando não efetua nenhum tipo de ordem nas linhas do arquivo, nem estabelece margens. Estas tarefas são executadas por comandos mais complexos e de mais potência como são o **nroff** e **troff**, cuja discussão escapa aos propósitos deste livro.

O comando **pr** não produz nenhum tipo de mensagem de erro no caso de não encontrar os arquivos a imprimir ou que se digitem opções incorretas. Existe outro detalhe acerca do comando. Quando se começa a imprimir a listagem no terminal, anulam-se todas as comunicações estabelecidas através do comando **write**, de tal forma que se evita a alteração do formato da listagem. Vejamos as opções mais comuns do comando:

- -<n> a saída é impressa no formato de "n" colunas;
- +<n> imprime o arquivo a partir da linha "n" especificada, inclusive;
- -h<cadeia> indica mudança do início. A <cadeia> é agora o início de cada página;
- -w<n> a largura de cada página da listagem será de "n" caracteres, em lugar de 72, tomado por default;
- -l<n> troca o número de linhas por página ao valor "n", ao invés de 66 linhas tomadas por default;
- -t a listagem evita (salta) as 5 linhas do início e do fim da página;
- -s<caracter> ao invés de separar as colunas pelo caracter de tabulação, estas são separadas pelo caracter especificado no <caracter>;
- -m imprime todos os arquivos especificados em colunas paralelas e simultaneamente;

Mudança de nome, cópia e eliminação de arquivos

Uma das tarefas mais normais no trabalho com um sistema

operacional é a mudança do nome de um determinado arquivo. O comando que o efetua denomina-se **mv** e sua sintaxe é:

mv < nome antigo> <nome novo>

```
$ mv texto arquivo
$ ls
arquivo
arq.1
$ cat texto
cat: can't open texto
$
```

Trocamos o nome do arquivo "texto" para "arquivo". O conteúdo do arquivo permanece inalterado. Posteriormente, executamos o comando **ls**, mostrando-nos uma lista dos arquivos do usuário na qual não aparece o arquivo "texto" mas sim "arquivo". Se tentarmos listar o conteúdo do arquivo "texto" através do comando **cat**, este nos avisará com uma mensagem de erro indicativo de sua inexistência. Leve muito em conta que, se pretendermos mudar o nome de um arquivo por outro que já exista, este será eliminado do sistema e passará a conter a informação do arquivo cuja alteração do nome foi efetuada.

Na sessão dedicada aos diretórios UNIX, voltaremos sobre este comando para ver a possibilidade de "mover" arquivos dentro dos caminhos (pathnames) da árvore de diretórios do sistema e do usuário.

Para efetuar cópias de arquivos, isto é, efetuar duplicação dos mesmos com nomes diferentes, existe o comando **cp**, que mantém várias versões de um determinado arquivo:

```
$ cp arquivo salva
$ cat salva
Texto do documento uno
$ cp arq.1 arq.2
$ cat arq.2
O sistema UNIX em estudo
$
```

Foram geradas duas cópias, uma do arquivo "arquivo" sobre outro de nome "salva" e outra do arquivo "arq.1" sobre "arq.2", e que possuirão exatamente o mesmo conteúdo, como vimos através do comando **cat salva** e **cat arq.2**.

```
$ cp arq.1 files
cp: can't open arq. 1
$ cp arquivo arquivo
cp: cannot copy file to itself
$ cp -s arquivo
Usage: cp f1 f2 or cp fl..fn d2
```

Pudemos ver que o comando **cp** avisa no caso de não encontrar o arquivo que irá ser copiado. Igualmente, no caso de tentar copiar um arquivo sobre si mesmo, nos avisa de tal situação. Por último, o comando não possui nenhum tipo de opções e, no caso de digitar alguma, nos avisa de que não é correta a sintaxe do comando.

Existe outro comando que permite a eliminação de arquivos do usuário ou sistema, de nome **rm** e cuja sintaxe é:

rm <lista de arquivos>

```
$ rm arq. 2
$ rm texto
rm: texto non existent
$ rm salva
$
```

Devemos ter muito cuidado com os arquivos que serão eliminados, visto que o comando **rm** não solicita a confirmação para tal eliminação. Existe uma opção do comando, de nome -i, que solicita interativamente a confirmação para cada um dos arquivos da lista de argumentos do comando. Esta opção deve ser usada quando não se sabe exatamente a lista dos arquivos a serem eliminados, por utilizar caracteres de máscara. A eliminação de arquivos é efetuada de "forma silenciosa", isto é, não é visualizado nenhum tipo de mensagem que avisa de que a eliminação foi executada. Vejamos as opções do comando **rm:**

rm [-rif] <arquivos>

- -f serão eliminados também aqueles arquivos protegidos contra escrita.
- -r elimina recursivamente todos os arquivos de um diretório e inclusive o próprio diretório.
- -i solicita confirmação para a eliminação de cada arquivo digitado.

Mensagens de erro do comando:

```
$ rm arq.1
rm: arq.1 non existent
$ rm -v arq.1 arq.2
rm: unknown option -v
$
```

Devemos ter muito cuidado com a opção -r, visto que eliminará sem pedir confirmação, todos os arquivos do diretório atual

e o próprio diretório. Neste caso, é muito conveniente usar a opção -i para solicitar a confirmação de cada arquivo a ser eliminado.

Normas adicionais sobre nomes de arquivos

Vamos destacar algumas normas para que os nomes dados aos arquivos sejam, de alguma forma, corretos ao sistema e ao usuário. Em primeiro lugar, o comprimento máximo do nome de um arquivo é de quatorze (14) caracteres. Se ultrapassarmos este comprimento, o sistema trunca o nome nos quatorze primeiros caracteres. Em segundo lugar, os caracteres que compõem o nome do arquivo podem abranger uma grande parte do jogo de caracteres do sistema, porém o sentido comum nos faz pensar em eliminar uma série deles que podem dar lugar a confusões para o usuário e o sistema. Por exemplo, se dermos a um arquivo o nome "-t", ao tentar localizá-lo através do comando **ls-t**, o sistema interpretará que se deseja uma lista de todos os arquivos do usuário classificada em ordem de tempo da última modificação e não que é o arquivo concreto desejado pelo usuário. Podemos deduzir que nunca devemos atribuir como primeiro caracter do nome de um arquivo o "-". Para tentar evitar todos estes problemas lhes aconselhamos somente sejam usados caracteres alfabéticos, numéricos e os símbolos "." (ponto) e "-" (sublihado) na composição dos nomes de arquivos, até que o usuário esteja completamente seguro da situação que se possa criar na atribuição de nomes aos mesmos.

Comandos de processamento de arquivos

Uma vez que vimos a forma de criar arquivos, localizá-los e listar seu conteúdo, vamos desenvolver uma série de comandos de uso geral e que oferecem uma potente ferramenta de trabalho sobre arquivos. Para a discussão dos mesmos, vamos criar um arquivo com um determinado conteúdo

```
$ ed
a
Com cem canhões por lado
vento em popa a toda vela
Não cruze o mar senão voa
```

```
um veleiro pequeno.
Poema vento.
.
w poema
114
q
$
```

O primeiro comando que vamos desenvolver efetua a conta do número de linhas, palavras e caracteres que compõem um ou nais arquivos Seu nome é **wc** (wc rd counter):

```
$ wc poema
5       22      114     poema
$
```

Uma palavra pode ser definida como o conjunto de caracteres que não contém nenhum branco, tabulador ou "newline". O arquivo "poema" contém 5 linhas, 22 palavras e 114 caracteres. No caso de que ao comando **wc** lhe sejam passados como argumentos uma lista de arquivos (mais de um), nos listará a conta para cada um dos arquivos da lista, assim como o número total de caracteres, palavras e linhas na totalidade dos mesmos:

```
$ wc poema arquivo
   5              22              114             poema
   1               4               23             arquivo
   6              26              137             total
$
```

Vejamos as opções do comando:

wc [-lwc] <arquivos>

- -l conta unicamente as linhas do arquivo
- -w conta unicamente as palavras do arquivo
- -c conta unicamente os caracteres do arquivo

Mensagens de erro do comando:

```
$ wc -n arquivo
arquivo
$ wc -lv arquivo
      1 arquivo
$ wc +c arquivo
wc: can't open +c
      1       4       23      arquivo
$
```

No primeiro caso, ao ser inválida a opção e existir o arquivo, não mostra nenhum tipo de informação na conta do mesmo, no segundo caso ignora a opção inválida e mostra a informação correspondente à opção válida; no terceiro caso nos avisa da inexistência do arquivo.

Outro importante comando denomina-se **grep**. Encarrega-se de buscar dentro dos arquivos que lhe são passados como argumentos, as linhas que contenham uma cadeia determinada que dá ao comando como parâmetros. Isto é, passada uma determinada cadeia de caracteres, visualizamos as linhas do arquivo (s) que contém tal cadeia:

```
$ grep vento poema
  vento em popa a toda vela
Poema vento
$ grep documento arquivo
Texto documento uno
$
```

Sobre o arquivo "poema" localizamos todas as linhas que contêm a cadeia "vento", e sobre o arquivo "arquivo", as que contêm "documento". O comando **grep** também localiza as linhas que "não" contêm a cadeia especificada. Isto se consegue através da opção **-v**:

```
$ grep -v vento poema
Com cem canhões por lado
Não cruze o mar senão voa
um veleiro pequeno.
$
```

O comando **grep** possui numerosas opções que efetuam tarefas diferentes baseadas na norma fundamental do comando, que é a busca de linhas de arquivos que contenham uma determinada cadeia de caracteres. Pode-se efetuar a busca em vários arquivos, contar linhas e caracteres e numerar linhas. Vejamos as opções mais comuns e importantes do comando:

grep <opções> <cadeia> <arquivos>

- -v Mostra todas as linhas exceto as que contenham a <cadeia> dada.
- -c Mostra todas as linhas que contenham a <cadeia> digitada.
- -l Mostra os nomes dos arquivos que contenham a <cadeia> digitada.

- -n Além de mostrar as linhas, visualiza o número da linha.
- -h Não visualiza o nome dos arquivos.
- -y Dá igual tratamento às maiúsculas e às minúsculas.
- -e Permite comentar <cadeia> pelo caracter "-".

Outro comando de grande importância é **sort**, que executa uma classificação das linhas do arquivo de entrada em ordem semi-alfabética. Isto é, para classificar duas linhas, compara o primeiro caracter de cada uma, se são iguais ao segundo, e assim até que exista uma diferença que permita estabelecer a classificação entre ambas:

```
$ sort poema
  um veleiro pequeno.
  vento em popa a toda vela
Com cem canhões por lado
Não cruze o mar senão voa
Poema vento
$
```

A classificação é linha por linha. A ordem natural de classificação inclui antes os brancos, depois as letras maiúsculas e, por último, as minúsculas, portanto não é estritamente uma classificação alfabética. O comando **sort** possui múltiplas opções para controlar a ordem da classificação: possibilidade de ordem inversa, numérica, ignorando brancos repetidos, classificando campos dentro da linha, etc. Vejamos a seguir as opções mais comuns e importantes do comando:

sort <opções> <arquivos>

- -r Inverte a ordem normal de classificação.
- -n Classifica em ordem numérica.
- -nr Classifica em ordem numérica inversa.
- -f Considera igual as maiúsculas e minúsculas.
- + n Começa a classificar no campo n + 1.

Outro interessante comando para visualizar conteúdos de um arquivo é **tail**. Este comando visualiza as últimas dez (10) linhas de um arquivo. Em arquivos como "poema" não há interesse, porém em arquivos maiores pode ser muito importante. O comando **tail** possui opções para especificar o número da linha a ser visualizada, de tal forma que podemos selecioná-las:

```
$ tail arquivo
Texto documento uno
$ tail -1 poema
Poema vento
$ tail +3 poema
Não cruze o mar senão voa
um veleiro pequeno.
Poema vento
$
```

Através da opção - <n> , onde <n> deve ser um dado numérico, visualizamos as "n" últimas linhas do arquivo. Neste caso, executamos **tail -1 poema**, que nos mostrará a última linha do arquivo "poema". A opção **+** <n> onde <n> tem o mesmo significado anterior, visualiza a partir da "n" linha do arquivo. O comando executado **tail +3 poema** mostra todas as linhas do arquivo "poema" a partir da terceira, inclusive. Vejamos as opções do comando:

tail [+ -] <n> <opções> <arquivo>

\+ Indica começo da listagem a partir da <n> linha do arquivo, inclusive.
Indica o começo da listagem a partir de <n> linhas do final do arquivo.

OPÇÕES

- l O número <n> refere-se a linhas. Esta opção é por default.
- c O número <n> refere-se a caracteres.
- b O número <n> refere-se a blocos de arquivos (grupos de 512 caracteres).

Os últimos dois comandos que vamos desenvolver permitem verificar comparações e diferenças entre arquivos. Para isto vamos criar outro arquivo:

```
$ ed
a
Com cem canhões por lado
vento em popa a toda vela
Não cruze o mal senão voa
  um veleiro pequeno
Poema vento
•
w poema • dois
119
q
$
```

Vimos que entre ambos os arquivos não existe muita diferença, somente duas palavras (cânones e mal) são diferentes. O comando **cmp** encontra a primeira linha na qual diferem dois arquivos e nos indica a posição de diferença entre ambos.

```
$ cmp poema poema.dois
poema poema. dois differ: char 13, line 1
$
```

O comando cmp nos indica que os dois arquivos são idênticos até o caracter 13 da primeira linha. Não nos informa as diferenças entre ambos, mas somente que está na primeira linha. Este comando é muito interessante para casos em que desejamos verificar a igualdade entre dois arquivos.

Existe outro comando de nome **diff** que nos dá informação das linhas que são diferentes, as que faltam e as que sobram, na comparação de ambos os arquivos:

```
$ diff poema poema.dois
1c1
<    Com cem canhões por lado
---  ---  ---  ---  ---
>    Com cem canhões por lado
3c3
<    Não cruze o mar senão voa
--   -  -  -  -  -  -  -  --
>    Não cruze o mal senão voa
$
```

O comando nos indica que a linha 1 do primeiro arquivo ("poema") está mudada em relação à linha 1 do segundo ("poema.dois") e igualmente com a terceira.

O comando **cmp** funciona com múltiplos tipos de arquivos, enquanto que o **diff** somente funciona com arquivos do tipo texto. O comando **diff** é usado quando desejamos conhecer as diferenças exatas entre dois arquivos. Evidentemente, é mais ápido o comando **cmp**, pois somente busca a primeira diferença. Vejamos as opções do comando **diff**:

diff [-fbhe] <arquivo1> <arquivo2>

- -f a saída das diferenças entre os dois arquivos é efetuada em ordem inversa.
- -h localiza pequenas diferenças, porém atua de forma muito mais rápida, e é muito convenienter em grandes arquivos. Não é compatível com ed.
- -b Ignora os tabuladores e brancos do final de linhas, e trata como um só branco os repetitivos.

- Admite as ordens de acrescentar, eliminar e modificar linhas, com os mesmos comandos que o editor ed.

Mensagens de erro:

```
$ diff arquivo arq.1
diff cannot open arq.1
$ diff -x arquivo texto
$
```

O comando **diff** avisa no caso de algum dos dois arquivos não existir. No caso de digitarmos uma opção inválida, o comando não nos mostra nenhuma mensagem de erro.

TABELA DE COMANDOS mais comuns no manejo de Arquivos

ls	Lista nomes de todos os arquivos do diretório atual do usuário.
ls <arquivos>	Lista somente os nomes da lista.
ls -t	Classifica em ordem de data-hora da última modificação dos arquivos.
ls -l	Informação completa de todos os arquivos. Também ls -lt.
ls -u	Classifica por data-hora da última utilização dos arquivos. Também ls -lu e ls -lut.
ls -r	Inverte a ordem de saída. Também ls -rt, ls -rlt, etc.
ed <arquivo>	Edição do arquivo indicado.
cp <arq.1> <arq.2>	Copia <arq.1> sobre <arq.2> . Copia em cima do <arq.2> se este já existe.
mv <arq.1> arq.2>	Move ("renomeia") <arq.1> ao <arq.2> . Copia sobre <arq.2> se este já existe.
rm <arquivos>	Elimina arquivos indicados irrevogavelmente.
cat <arquivos>	Lista (visualiza) o conteúdo dos arquivos indicados.

TABELA DE COMANDOS mais comuns no manejo de Arquivos	
pr <arquivos>	Lista o conteúdo dos arquivos com páginas de 66 linhas e cabeçalho
pr -n <arquivos>	Igual, porém em formato de "n" colunas.
pr -m <arquivos>	Igual, porém em múltiplas colunas. Um arquivo ao lado do outro.
wc <arquivos>	Conta linhas, palavras e caracteres em cada arquivo e em seu total.
wc -1 <arquivos>	Conta somente linhas
grep <cad> <arquivos>	Lista linhas dos arquivos que contenham a cadeia <cad>.
grep -v <cad> <arquivos>	Lista linhas dos arquivos que não contenham a cadeia <cad>.
sort <arquivos>	Classifica os arquivos em ordem alfabética de linhas.
tail <arquivo>	Lista as últimas 10 linhas do arquivo.
tail -n <arquivo>	Lista as "n" últimas linhas do arquivo.
tail + n <arquivo>	Lista as linhas do arquivo começando na linha "n".
cmp <arq.1> <arq.2>	Indica a primeira diferença localizada entre os dois arquivos.
diff <arq.1> <arq.2>	Lista todas as diferenças entre os dois arquivos.

Diretórios

O sistema operacional UNIX distingue a cada um dos arquivos do sistema, por seu nome, pelo agrupamento dos mesmos dentro de entidades denominadas diretórios. É similar à forma em que os livros são colocados em uma biblioteca dentro de estantes. A

biblioteca é o sistema, as estantes são os diretórios e os livros são os arquivos.

Cada usuário do sistema (definido entre outras coisas por seu nome) tem um diretório particular, denominado diretório do usuário ou diretório corrente. O usuário pode estar trabalhando em outro diretório, mas sempre possuirá o seu próprio. A menos que se efetue uma mudança de diretório, qualquer arquivo criado pelo usuário pertencerá ao diretório dele mesmo. Ao efetuar a conexão ao sistema (login) é produzida a entrada no diretório do usuário que posteriormente veremos como se representa.

Um diretório, além de possuir arquivos normais, pode conter outros diretórios. A forma normal de representação desta situação é a de uma árvore de arquivos e diretórios. O sistema dá a possibilidade de mover-se dentro desta árvore e localizar qualquer arquivo do sistema iniciando a busca na raiz **(root)** da árvore. O comando básico de localização do diretório atual no qual o usuário está trabalhando denomina-se **pwd.**

```
$ pwd
/usr/provas
$
```

O comando visualiza o diretório atual de trabalho, que coincide com o diretório do usuário "provas". Como podemos ver, o diretório do usuário é um diretório dentro do diretório **usr**, que por sua vez é um diretório do diretório base da árvore, denominado **(root) (raiz)** e que é representado por "/". Este caracter é usado como separador dos diretórios e arquivos da árvore. Também existe a limitação de 14 caracteres para o nome de um diretório. Em quase todos os sistemas UNIX, os usuários do mesmo ramo da árvore na forma **/usr**. Vejamos alguns exemplos de comando **ls** incorporando a noção de diretórios:

```
$ ls/usr/provas
arquivo
arq.1
$ ls /usr
antonio
diretor
fontes
maestro
provas
$ ls/
bin
boot
dev
etc
lib
tmp
```

```
unix
usr
$
```

O comando **ls /usr/provas** lista os arquivos que existem no usuário "provas", isto é, dentro de seu diretório normal. O comando **ls /usr** lista os nomes dos diretórios de usuários definidos no sistema ou, em geral, a lista de arquivos e diretórios que ramificam da raiz **(root)** da árvore. Vejamos alguns exemplos com o comando **cat:**

```
$ cat /usr/provas/arquivo
Texto do documento uno
$ cat /usr/diretor/prova
prova de textos.
arquivo de prova
$ ls /usr/diretor
prova
carta
$
```

Representamos um arquivo pelo caminho que este percorre dentro da árvore do sistema, desde a raiz do mesmo (root). Este caminho denomina-se no UNIX **pathname** e seu significado é o percurso que se deve fazer na árvore do sistema para localizar o arquivo, partindo da base (root) da árvore. Como vimos, foi listado o conteúdo de um arquivo "prova" residente no usuário "diretor". É uma norma universal do UNIX poder representar um arquivo por seu caminho ou "pathname". Assim mesmo, é possível listar os nomes de arquivos de outro usuário do sistema através de seu caminho (pathname) da árvore. A estrutura da árvore que reside em um sistema UNIX com vários usuários, poderia ser a representada na figura 1.

Vamos executar o comando **cp**, definindo arquivos por seu caminho (pathname) na árvore de diretórios do sistema:

```
$ cp /usr/diretor/prova dados
$ ed /usr/diretor/prova
....
....
$
```

Como pudemos ver, efetuamos a cópia de um arquivo de usuário "diretor" sobre o arquivo "dados" no diretório onde o usuário está trabalhando (normalmente **/usr/provas**). Assim mesmo, editamos o próprio arquivo do usuário "diretor". Chegando a este ponto, vamos desenvolver um tema importante no UNIX: as permissões de acesso a arquivos do sistema.

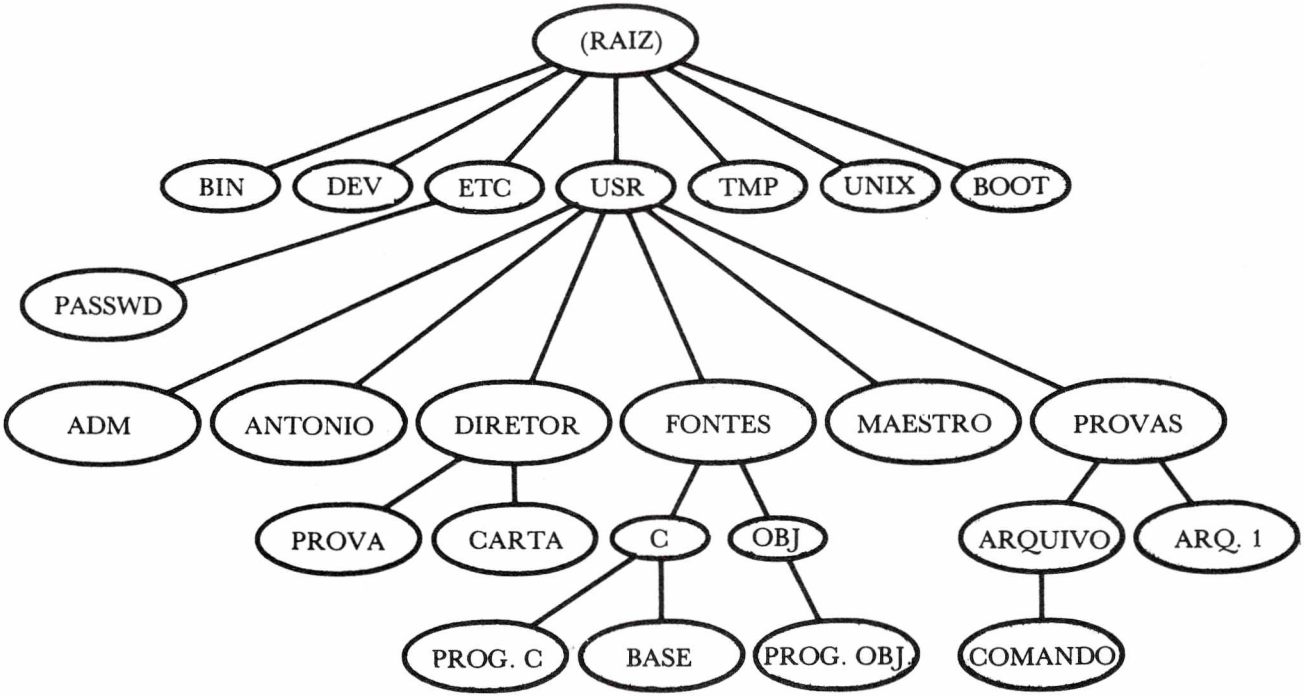

Figura 1. — Possível estrutura da árvore de um sistema UNIX com vários usuários.

Permissões de acesso aos arquivos

Cada um dos arquivos residentes no sistema possui um conjunto de permissões de acesso para os usuários do mesmo. O usuário pode mudar as permissões de acesso de "seus" arquivos, de tal forma que proteja a informação de alguma maneira. Como comentamos em outro capítulo, existe um superusuário do sistema, que pode voltar a mudar nossas permissões, de forma que desproteja e tenha acesso a nossa informação.

O sistema reconhece cada usuário por um certo número que lhe é atribuído na hora de sua criação. O nome do usuário não é mais que um identificador para o sistema e uma espécie de chave de acesso ao mesmo. Devido a isto, vários nomes de usuários diferentes podem ter o mesmo número identificador. Esta situação não é nada segura dentro de um sistema e o administrador do mesmo se encarregará de controlá-la. Cada usuário, além de seu número de identificação, possui um número identificador do grupo ao qual pertence. Em muitos dos sistemas UNIX, o grupo de todos os usuários do sistema denomina-se **other**. Toda a informação referente aos parâmetros dos grupos e usuários do sistema encontra-se em um arquivo localizado no pathname **/etc/passwd**.

Existem três tipos de permissões de acesso a um arquivo do sistema:

- read **r** (leitura)
- write **w** (escrita)
- execute **x** (execução)

A opção -l do comando ls nos mostra as permissões de acesso que possui o arquivo que é visualizado:

```
$ ls -l /etc/passwd
-rw-r--r-- 1 root      2048   May   9  13:50   /etc/passwd
$ ls -lg /etc/passwd
-rw-r--r-- 1 adm       2048   May   9  13:50   /etc/passwd
$ ls -l /bin/passwd
-rwxr-xr-x 1 root      8484   Feb  24  10:00   /bin/passwd
$ ls -l /bin/who
-rwxrwxr-x 1 root      6348   Mar   1  08:00   /bin/who
$ ls -l arquivo
-rw-r--r-- 1 provas      24   May  17  18:30   arquivo
$
```

A opção -l do primeiro comando nos informa o nome do proprietário ou usuário do arquivo, que corresponde ao nome do usuá-

rio **root**. A opção **-g** nos mostra o nome do grupo do arquivo, neste caso **adm**. A cadeia **-rw-r--r--** nos dá informação acerca das permissões dos arquivos. O primeiro caracter da cadeia (-) indica o tipo de arquivo do qual se trata; neste caso, um arquivo ordinário. No caso de ser um diretório, apareceria o caracter "**d**"

Os três caracteres seguintes da cadeia indicam os acessos autorizados do arquivo para o próprio usuário dele. As três seguintes são para o **grupo** de usuários ao qual pertence o próprio usuário criador do arquivo e as três últimas posições correspondem aos acessos para o resto de usuários do sistema. Vejamos um quadro de interpretação destes temas:

ACESSO AOS ARQUIVOS	
-	Indica que NÃO há permissão para acessar ao arquivo.
r	Permite-se acesso de leitura ao arquivo.
w	Permite-se acesso de escrita no arquivo.
x	Permite-se a execução do arquivo. Contém um processamento executável pelo sistema.
Cadeia de permissões de acesso "tabdefghi"	
	posições 1234567890
t → posição 1	tipo de arquivo: - ordinário **d** diretório.
a → posição 2	leitura para o usuário: - não permitida **r** permitida.
b → posição 3	escrita para usuário: - não permitida **w** permitida.
c → posição 4	execução para usuário: - não permitida **x** permitida.
d → posição 5	leitura para o grupo de usuários.
e → posição 6	escrita para o grupo de usuários.
f → posição 7	execução para o grupo de usuários.
g → posição 8	leitura para o resto de usuários.
h → posição 9	escrita para o resto de usuários.
i → posição 0	execução para o resto de usuários.

Exemplos:

-rw-r--r-- → arquivo ordinário, Permitida leitura e escrita ao usuário do arquivo. Permitida ao grupo e resto de usuários. Não permitida escrita ao

grupo e resto de usuários. Não permitida execução a nenhum usuário do sistema.

-rwxr-xr-x → arquivo ordinário. Permitida leitura e execução a todos os usuários, usuário próprio, grupo e resto de usuários. Permitida escrita somente ao usuário do arquivo.

-rwxrwxr-x → arquivo ordinário. Permitida leitura e execução a todos os usuários do sistema. Permitida escrita ao usuário e ao grupo. Não permitida a escrita ao resto de usuários.

Quando enviamos à Shell a ordem de execução de um comando, por exemplo **who**, esta busca em uma série de diretórios do sistema até encontrar tal arquivo. Um dos diretórios onde foi realizada a busca é **/lib**, onde encontra-se o arquivo **who**. O núcleo ou **kernel** do sistema UNIX comprova as permissões de acesso ao arquivo e opera em conseqüência. Se este comando possui permissão de execução para o usuário que deseja executar, o sistema validará a execução; em caso contrário não possuindo permissão para execução, nos avisará com uma mensagem e não executará o comando.

As permissões de acesso a diretórios têm igual estrutrura, porém se diferenciam em alguns aspectos:

```
$ ls -ld.
drwxrwxr-x      3       provas      50 May 12 23:00.
$
```

Como vimos, incluímos a opção **-d** dentro do comando **ls**. Esta opção visualiza a informação de um diretório (a anotação) "." correspondente à nomenclatura do próprio diretório do usuário. A proteção **r** indica que o diretório tem permissão de leitura. A proteção **w** indica que podemos criar e eliminar arquivos dentro do diretório. A permissão para eliminar um arquivo dentro de um diretório é independente do próprio arquivo. Se o usuário possui uma permissão **w** em um diretório, pode eliminar arquivos do mesmo, inclusive se estiverem protegidos. O comando **rm** solicitará confirmação para eliminar os arquivos protegidos. A proteção **x** em um diretório indica que se possui acesso de busca de arquivos dentro do diretório. Isto é, que se um usuário possui permissão de execução em um diretório, pode buscar arquivos através dele mesmo. Com estas premissas, pode-se assimilar as tarefas que efetuam as proteções de acesso aos diretórios.

Vamos explicar, sem entrar em excessivos detalhes e por meio de uma tabela, o comando **chmod**, que efetua a mudança de permissões de acesso nos arquivos:

COMANDO CHMOD	
chmod < quem > < operação > < permissão > < arquivos >	
< quem >	
u	usuário próprio do arquivo
g	grupo de usuários associados ao usuário
o	resto de usuários do sistema
a	todos (usuários, grupo e resto). Opção por default
< operação >	
+	acrescenta permissão digitada
-	tira permissão digitada
=	supõe permissão absoluta para os arquivos
< permissão >	
r	permissão para leitura
w	permissão para escrever
x	permissão para execução

Exemplos:

```
$ ls -l exemplo
-rwxrwxrwx     .....
$ chmod go-wx exemplo
$ ls -l exemplo
-rwxr--r--     .....
$ chmod g + x exemplo
$ ls -l exemplo
-rwxr-xr-x     .....
$ chmod a = w exemplo
$ ls -l exemplo
-rwxrwxrwx     .....
$ chmod r-x exemplo
$ ls -l exemplo
-rwxrwxrw-     .....
```

Existe outra forma de manejo do comando, mais complicada, que deixamos a critério do leitor consultá-la nos manuais originais do UNIX.

Vamos desenvolver o comando **cd**, que permite o movimento do usuário pelos diretórios próprios ou pelos do sistema. O comando **cd** posiciona o usuário no diretório indicado com uma certa nomenclatura:

cd	troca o diretório normal do usuário:/usr/ <nome usuário> troca o diretório especificado pelo <percurso>, que deve ser um caminho (pathname) completo.
cd <percurso>	

Exemplos:

```
$ cd
$ pwd
/usr/provas
$ cd /usr
$ pwd
/usr
$ cd /usr/provas/programas
$ pwd
/usr/provas/programas
```

cd <sub-diretório>	troca o sub-diretório indicado. Não faz falta colocar o percurso completo. Deve ser um diretório dentro do atual no qual se está.
cd ..	sobe um nível na estrutura (hierárquica) da árvore.
cd .	mantém-se no diretório atual.

Exemplos:

```
$ cd
$ cd programas
$ pwd
/usr/provas/programas
$ cd ..
$ pwd
/usr/provas
$ cd arquivo
arquivo: bad directory
$ cd
$ cd .
$ pwd
/usr/provas
$
```

Vejamos dois comandos que permitem a criação e eliminação de diretórios. O comando **mkdir** (criação de diretórios) e o comando **rmdir** (eliminação de diretórios).

```
$ cd
$ mkdir sub
$ cd sub
$ pwd
/usr/provas/sub
$ mkdir fontes textos
$ cd fontes
$ pwd
/usr/provas/sub/fontes
$ cd ..
$ pwd
/usr/provas/sub
$
```

O comando **mkdir** cria os diretórios dentro do diretório atual de trabalho do usuário. Para criar um diretório deve-se possuir permissão de escrita no diretório onde se encontra o usuário, isto é, no diretório pai do qual será criado.

```
$ cd
$ cd sub
$ ls -d
fontes textos
$ rmdir textos
$ ls -d
fontes
$ cd .
$ pwd
/usr/provas
$ rmdir sub
rmdir: sub not empty
$
```

No caso em que um diretório contenha algum arquivo ou diretório o comando **rmdir** não permitirá sua eliminação e nos apresentará uma mensagem de aviso. A eliminação de um diretório, igual ao do arquivo, é aceita pelo comando se o usuário possuir permissão de escrita no mesmo.

Busca de arquivos em diretórios

Existe um comando UNIX que permite a busca ou localização de um arquivo dentro da árvore de diretórios. Seu nome é **find**. As especificações que podemos dar para a busca do arquivo incluem possibilidades desde o nome do arquivo, proprietário, nome do grupo ao qual pertence, ligações (links) e, inclusive, data-hora da última modificação ou acesso ao mesmo.

find < diretórios > < condições busca >

busca a partir dos diretórios especificados, os arquivos que cumpram as condições dadas.

A lista de diretórios deve estar separada por brancos ou tabuladores, e efetua a busca a partir desses diretórios e seus sub-diretórios.

Condições:

name < arquivos >	busca os arquivos especificados
-type < df >	d→arquivo tipo diretório; f→arquivo totalmente cheio:
-links < n >	busca arquivos com < n > ligações;
-user < usuário >	busca os arquivos correspondentes ao < usuário > especificado;
-group < grupo >	busca os arquivos correspondentes ao grupo especificado;
-size < n >	busca arquivos com tamanho de < n > blocos (512 bytes)
-atime < n >	busca arquivos nos quais há < n > dias foi efetuado um acesso;
-mtime < n >	busca arquivos nos quais há < n > dias foi efetuada uma modificação;
-print	imprime o caminho percorrido dentro da árvore de diretórios do sistema.

CAPÍTULO V

O INTÉRPRETE DE COMANDOS ("SHELL") DO SISTEMA

Uma vez realizada a conexão ao computador por meio do comando "login", o sistema devolve um "prompt" indicando que se encontra preparado para receber uma ordem do terminal. O caracter de "prompt" consiste em um sinal "$" para usuários normais ou um " # " para o super-usuário ou ou "root".

Do ponto de vista do usuário, o intérprete de comandos proporciona um meio simples de comunicar-se com o sistema, oferecendo-lhe duas possibilidades:

a) Executar diretamente qualquer comando do computador.
b) Escrever e executar instruções pertencentes à própria linguagem do intérprete de comandos.

Entre as características mais importantes do intérprete de comandos, chamado "Shell" ("concha" ou "couraça" que isola e protege o usuário do sistema), podemos citar:

- Execução direta de comandos do sistema operacional, transpasse de opções e argumentos, e substituição de variáveis e cadeias de caracteres.
- Construções de alto nível como while, if-then-else, case e for.
- Modificação do ambiente de execução ("environment") do usuário (conjunto de variáveis e formas de funcionamento pré-definidos).

- Reencaminhamento da Entrada/Saída para arquivos, e comunicação de processamentos através de "pipes" ou tubulações.

Comandos simples

Os comandos simples consistem em uma ou mais palavras separadas por espaços em branco. A primeira palavra é o nome do comando a ser executado e as seguintes são as opções e argumentos do comando. Por exemplo:

date<RETURN>

executaria o comando "date", devolvendo-nos como resultado a data e hora do sistema em um formato como:

Thu May 15 21:15:05 GMT + 1:00 1986

Indicando que se trata de quinta-feira 15 de maio de 1986, são 21 horas, 15 minutos e 5 segundos.

Analogamente, o comando

ls - l

produziria uma listagem do diretório (comando ls) em formato estendido (opção - l).

Para proceder à excecução de um comando, normalmente a Shell cria um novo processamento e espera até a sua finalização. Uma linha como

cc nome.c

chamaria o compilador da linguagem C para proceder a compilação ao programa fonte "nome.c", devolvendo o prompt "$" da Shell uma vez finalizada a compilação do programa.

Ao contrário

cc nome.c &

executaria a compilação do programa em "background" (de forma não interativa), de maneira que a compilação do programa se-

ria produzida por meio de um novo processamento criado nesse momento, devolvendo-se o controle ao terminal imediatamente, sem esperar que a compilação finalizasse.

O operador "&" indica à Shell que não deve esperar o término de um processamento. Para permitir o controle do mesmo por meio de comandos como "ps" ("process.status", mostra o estado de um ou vários processamentos) ou "Kill" (força o término de um processamento), a Shell devolve seu número de processamento ou "pid" (process indentifier).

Reendereçamento da entrada/saída

A quase totalidade dos comandos do UNIX enviam sua saída ao arquivo de saída padrão, correspondente em princípio ao terminal, ainda que esta correspondência possa modificar-se e conseguir o reencaminhamento da saída para qualquer outro arquivo. Por exemplo, a execução do comando

```
ls-l > arquivo
```

não produziria nenhuma saída sobre o terminal, enviando o resultado da execução do comando ls-l (listagem de um diretório em formato estendido) ao arquivo denominado "arquivo".

O caracter " > " indica um reendereçamento da saída padrão. Se o arquivo indicado para a saída não existir será criado pela Shell e, pelo contrário, se já existir, seu conteúdo será resituado pela saída do comando; se o arquivo não fosse acessível ou não fosse possível criá-lo, obteríamos uma mensagem de erro.

Analogamente, pode-se dirigir a entrada padrão por meio do caracter " < ", utilizado como:

```
wc < arquivo
```

Executando o comando "wc" ("word counter" ou contador de palavras), que conta o número de linhas, palavras e caracteres presentes no arquivo de entrada padrão, neste caso "arquivo", devido ao reendereçamento da entrada.

O reencaminhamento da saída padrão de um programa à entrada padrão de outro, é conseguido através do emprego dos "pipes" (" | ") ou tubulações. Assim:

```
ls | sort-r
```

produziria uma listagem do diretório classificado em ordem alfabética inversa: "ls" daria lugar a uma listagem do diretório sobre a saída padrão, que seria reendereçada pelo efeito do "Pipe" " " para a entrada do comando "sort", que através da opção "-r" produziria a classificação em ordem alfabética invesa.

É algo semelhante ao efeito dos comandos:

```
ls > auxiliar
sort -r auxiliar
```

com a diferença de que não é necessária a utilização do arquivo intermediário "auxiliar". Além disso, os dois comandos conectados através de um "pipe" são executados simultaneamente e não um em seguida do outro, como ocorreria neste segundo caso.

Os "pipes" não são bi-direcionais (para isto seria necessário a utilização de dois "pipes"), no sentido de que não permitem a leitura e escrita em ambos os sentidos, mas somente a escrita em um extremo e a leitura em outro, e realizam sua sincronização por meio da detenção dos comandos quando estes não tenham nada que ler ou escrever. Assim, se deteria a execução do comando "sort" quando não tivesse nada que ler e a do "ls" quando não tivesse nada mais que escrever.

No sistema operacional UNIX denominam-se "filtros" aos programas que atuam sobre a entrada padrão e a modificam de alguma maneira antes de devolvê-la à saída padrão. Um exemplo típico é o comando "grep", que mostra a sua saída àquelas linhas de entrada que contenham uma determinada palavra. A execução de

```
grep fatura movimentos
```

faria com que fossem listadas sobre o arquivo de saída padrão todas as linhas de um arquivo denominado "movimentos", que contivessem em algum lugar a palavra "fatura", atuando como um "filtro" das linhas de entrada.

Um "pipe" pode constar de mais de dois comandos, produzindo-se o reencaminhamento múltiplo de suas entradas e saídas consecutivamente. Por exemplo,

```
ls | grep carta | more
```

Produzirá como resultado uma listagem na tela com paginação (espera antes de passar a uma nova página) de todos os nomes

de arquivos que contenham em algum lugar a palavra "carta".

Para fianalizar, realizemos um pequeno resumo acerca do reendereçamento da entrada e saída:

- >fl saída padrão padrão (descritor do arquivo 1) é enviada ao arquivo "f1", criando-se se não existisse anteriormente.
- >> 1 a saída padrão é enviada ao arquivo "f1". Se já existe o arquivo então acrescneta-se o conteúdo anterior, em caso contrário, cria-se o arquivo.
- < f1 Toma-se a entrada padrão (descritor do arquivo O) à partir do arquivo "f1".
- << f1 Toma-se a entrada padrão à partir das linhas de entrada Shell até que se encontre uma linha que contenha a palavra "f1".
- > & num O descritor do arquivo "num" é duplicada por meio da função primitiva "dup", utilizando-se o resultado como arquivo de saída padrão.
- < & num A entrada padrão é duplicada à partir do descritor do arquivo "num".

Geração de nomes de arquivos

A Shell proporciona um mecanismo para obtermos nomes de arquivos coordenados a uma máscara de formato determinada. Por exemplo,

ls-l*.c

Produz uma listagem de todos os nomes de arquivos que possuem a terminação ".c" (habitualmente programas fonte na lingua gem C). O caracter "*" consiste em um formato que verifica qualquer caracter, inclusive o caracter nulo. As máscaras usadas são:

- * Verificado por qualquer cadeia de caracteres, inclusive o caracter nulo.
- ? Verificado por um único caracter.
- [...] Verificado por qualquer dos caracteres dentro dos colchetes ([]). Se aparecer um par de caracteres separados por um hífen "—", então refere-se à classe de caracteres compreendido entre os dois especificados.

Por exemplo

ls bloco?

produziria uma listagem do diretório com todos os nomes de arquivos que começassem pela palavra "bloco" e contivessem, em seguida, qualquer caracter (blocol, bloco9, bloco z, ...).

[a-m]*

Seria verificado por todos os nomes de arquivos que começassem por uma letra compreendida entre a "a" e a "m", inclusive.

O caracter "*" não inclui aqueles nomes de arquivos que começam por um ponto ".". Assim, se tivéssemos um arquivo com o nome ".profile", por exemplo, não seria mostrado através do comando

ls*

Porém sim o seria mediante

ls*

Caracteres especiais (metacaracteres)

Os caracteres que possuem um significado especial para a Shell, como " > ", " < ", "*", "?", " | " e "&" denominam-se metacaracteres. Qualquer caracter precedido pelo sinal " \ " interpreta-se também como um metacaracter, mudando seu significado. Assim:

echo \ ?

mostrará na tela um único caracter "?", enquanto que

echo \\

fará o mesmo com o caracter " \ ".

Com o fim de permitir que as cadeias dos comandos possam alcançar o comprimento suficiente, com todas suas opções, ignora-se o caracter de saltar linha " \ n" no meio de uma cadeia de caracteres.

Para proteger uma cadeia de caracteres da Shell e evitar que

possa ser interpretada como se contivesse metacaracteres, podemos rodear a cadeia por aspas simples " ", como no caso de:

cat "bloco?"

Que produziria uma listagem por tela (comando cat) do arquivo "bloco?", em lugar de referir-se aos nomes dos arquivos "bloco0", "bloco1", ... "blocoz".

Programação na linguagem da Shell

A Shell pode ser empregada para ler e executar comandos contidos em um arquivo. Por exemplo, a instrução

sh arquivo [arg1 arg2 ... argn]

produziria uma chamada à Shell para que lesse e executasse os comandos contidos no "arquivo", dizendo-se então que "arquivo" é um "arquivo de comandos".

Podem ser utilizados como argumentos na chamada ao arquivo de comandos, os "parâmetros posicionais" $1, $2, ... $9. exemplo, se o arquivo f1 contiver o texto

cc —o $1 $1, c

então

sh f1 nome

é equivalente a

cc —o nome nome.c

Onde vimos que o parâmetro posicional S1 foi substituído por "nome". Este exemplo serviria para compilar um programa fonte na linguagem C chamado "nome.c", deixando o programa compilado em "nome", em lugar de "a.out", como denominado por default ao compilador de C a sua saída.

Também pode converter-se um arquivo de comandos em "executável" somente mudando suas proteções de grupo e usuário. Assim

```
chmod +x f1
```

produz o efeito de mudar a proteção de execução do arquivo f1 para o usuário, convertendo-o em executável, de maneira que

```
f1 nome
```

é equivalente a

```
sh f1 nome
```

permitindo que possa intercambiar-se o emprego de arquivos de comandos da Shell e autênticos comandos do sistema operacional. A execução do arquivo de comandos f1 seria equivalente à execução direta de qualquer comando, criando-se em cada caso um novo processamento para proceder à sua execução.

A Shell proporciona duas variáveis reservadas para seu emprego com parâmetros posicionais, como são o número de argumentos de chamada, contido na variável "**$#**", e o nome do arquivo em execução, ou "**$0**". Também existe uma variável reservada "**$***", equivalente a todos os parâmetros posicionais com exceção do primeiro ("**$0**").

Sentenças de controle na Shell

É relativamente freqüente a execução da Shell em loops que incluam parâmetros posicionais $1, $2, ... e são executadas uma vez para cada parâmetro posicional. Por exemplo, pensemos num arquivo de comandos denominado "tel" que busque nomes num arquivo "/usr/agenda/telefones", contendo linhas como

```
"..."
joao 4731291
tomas 4167801
fernando 4353218
alfredo 4450153
"..."
```

Se o arquivo de comandos "tel" contivesse

```
for i
do
        grep $i /usr/agenda/telefones
done
```

```
tel joao
```

mostraria na tela aquelas linhas que contivessem a cadeia de caracteres "joao", no arquivo "/usr/agenda/telefones".

Do mesmo modo

```
tel joao fernando
```

imprimiria as linhas que contivessem o nome "joao", e a seguir as que contivessem o nome "fernando".

Em geral, o loop for apresenta-se como

```
for var in v1 v2 ...
do
        lista-de-comandos
done
```

Onde lista-de-comandos refere-se a um ou mais comandos simples separados por um salto de linha ou por um ponto e vírgula ";". "do" e "done" são separadores para delimitar o corpo do loop, "var" é uma variável da Shell e "v1", "v2", são alguns de seus possíveis valores.

Cada vez que se encontre um valor de "var" compreendido entre "v1", "v2", ..., será executado o conjunto de instruções compreendidas no corpo do loop for.

No caso de que "v1", "v2", ... sejam omitidos, então será executado o loop para cada um dos parâmetros posicionais, como se tratasse da variável "$*".

A sentença case

O comando "case" tem a forma geral

```
case var in
    v1) comando;;
    v2) comando;;
    "..."
esac
```

A Shell comprova seqüencialmente se o valor de "var" coin-

cide com "v1", "v2", ..., e no caso de coincidência, executa o(s) comando(s) correspondente(s) finalizando a execução do "case". Como o "*" é um valor que verifica todas as variáveis, pode-se utilizar para implementar um valor por default, incluindo-o justamente antes da sentença "esac", indicativa do final do loop case. Por exemplo.

```
case $# in
        1) cat >> $1 ;;
        2) cat >> $2 < $1 ;;
        *) echo  emprego: acrescentar [desde] a ;;
```

seria um comando para acrescentar (somar) um arquivo a outro. Utilizado como

```
soma f1
```

$ # valeria 1, copiando a entrada padrão ao final do arquivo f1 por meio do comando cat. Analogamente,

```
soma f1 f2
```

acrescenta o conteúdo do arquivo f1 ao final do arquivo f2. Pelo contrário, se o número de argumentos de chamada fosse diferente de 1 ou de 2, então o valor "*" seria alcançado por default, passando a ser impressa uma mensagem de erro.

Também podem ser empregados valores alternativos em cada um dos "v1", "v2", ..., empregando-se uma barra vertical " | ". Assim:

```
case $x in
        —S | —s) ...
        —N | —n) ...
esac
```

seria equivalente a

```
case $x in —[Ss]) ... —[Nn]) ... esac
```

Variáveis da Shell

A Shell maneja variáveis do tipo cadeia de caracteres. Seu nome consiste em uma combinação de letras (forçosamente o pri-

meiro caracter), dígitos e caracteres "—". A forma de declarar as variáveis é atribuindo-lhes um valor, como em

```
i = 100 x1__max = f200j  s1 = f1
```

que acrescentaria às variáveis "i", "x1__max" e "s1" os valores "100", "f200j" e "f1", respectivamente.

Para acedermos ao valor de uma variável, é necessário antepor um caracter "&" a seu nome. Por exemplo:

```
echo i
```

imprimiria literalmente o caracter "i", enquanto que

```
echo $i
```

imprimiria o valor da variável "i", neste caso 100.

É freqüente o emprego de variáveis da Shell para armazenar nomes de diretórios, facilitando sua utilização. Assim

```
d = usr/joao/programas/fontes/c
mv *.c $d
mv /etc/*.c $d
```

levaria os arquivos fontes em linguagem C contidos no diretório atual de trabalho, e no diretório /etc, ao diretório "/usr/joao/programas/fontes/c".

Também emprega-se a construção

```
echo ${user}
```

equivalente neste caso a

```
echo $user
```

mas é utilizada quando o nome da variável vem seguida por uma letra ou dígito, como é o caso de

```
tmp = /tmp/provas
ps a>${tmp}1
```

produziria o reencaminhamento da saída do comando ps ao arquivo "/tmp/provas1", enquanto que

```
ps a>$tmp1
```

Faria referência ao valor da variável "tmp1", completamente diferente do anterior.

As seguintes variáveis são inicializadas pela Shell ao proceder a execução de cada comando:

- $? Consiste no código de retorno (status) do último comando executado, devolvido como uma cadeia de caracteres decimais. Normalmente emprega-se o valor 0 após a execução de um comando satisfatoriamente e um valor diferente de zero para indicar a existência de algum problema em sua execução.
- $# É o número de parâmetros posicionais, em decimal.
- $$ Consiste no número identificador do processamento desta Shell, em decimal. É utilizado com freqüência para gerar nomes de arquivos temporais, com a garantia de que sejam únicos. Por exemplo,

```
sp  >  a  /tmp/provas$$
rm /tmp/provasSS
```

- $! É o número identificador do processamento do último comando executado em modo "background" (não interativo).

Algumas variáveis possuem um significado especial para a Shell, passando-se ao "environment" ou conjuntura de execução do usuário, de forma que sejam acessíveis desde outros programas.

- $MAIL Quando se maneja interativamente, a Shell busca esta variável antes de devolver seu prompt "$". Se o arquivo especificado na variável foi modificado na última consulta, a shell imprimirá a mensagem "You have mail" antes de passar a solicitar o comando seguinte. Normalmente, MAIL se inicia no arquivo "profile", existente no diretório de trabalho do usuário. Por exemplo,

```
MAIL = /usr/spool/mail/joao
```

Inicializaria MAIL ao valor de um nome de arquivo no qual armazenaria o correio manejado pelo comando "mail".

- $HOME É o argumento por default para o comando "cd" (mudança de diretório). Assim a utilização de "cd" sem argumentos é equivalente ao emprego de

  cd $HOME

  A variável HOME também costuma ser inicializada no arquivo de comandos "profile".
- $PATH Consiste em uma lista de diretórios nos quais devem ser buscados os comandos a executar. Cada vez que se procede à execução de um comando, a shell busca este comando nos diretórios (e na ordem) especificados pelo PATH ou, por default, em "/bin" e "/usr/bin". PATH costuma ser inicializado no arquivo "profile", consistindo em uma série de nomes separados pelo caracter dois pontos ":"

  PATH = :/usr/joao/bin:/usr/ucb:/bin:/usr:bin

- $PS1 é o prompt primário da shell, por default o caracter "$".
- $PS2 é o prompt secundário da shell, quando a introdução de um comando foi incompleta e deve prosseguir a entrada. Por default consiste no caracter "> ".

O comando test

O comando "test", ainda que não faça parte da shell, é utilizado para seu uso exclusivo. Consiste naavaliação de uma determinada condição, devolvendo um valor certo ou falso conforme o caso. Por exemplo:

test —f f1

Devolve o valor 0 se o arquivo f1 existir e um valor diferente de zero em caso contrário. Entre ascomprovações mais freqüentes; podemos citar:

```
test s              certo se a cadeia "s" não é nula.
test —f  arquivo    certo se "arquivo" existe.
test —r  arquivo    certo se "arquivo" pode ser lido.
test —w arquivo     certo se "arquivo" pode ser escrito.
test —d arquivo     certo se "arquivo" é um diretório.
```

Sentenças while e until

O loop "while" tem a forma geral:

```
while condição
do
    comando(s);
done
```

Onde a condição comprovada por while costuma incluir a execução de um comando "test". Cada vez se comprova a condição e esta é certa, executa-se o comando(s) incluído(s) no corpo do loop. Por exemplo,

```
while test $1
do
    "..."
    shift
done
```

É equivalente à construção

```
for i
do
    "..."
done
```

"shift" é um novo comando da shell que desloca os parâmetros posicionais $2, $3, ..., convertendo-os em $1, $2, ..., e perdendo o parâmetro $1 anterior.

O loop "until" é semelhante ao "while" com a diferença que a condição de permanência no loop é invertida: executa-se o loop enquanto que a condição seja falsa e finaliza a execução quando for certa.

A sentença if

A shell possui uma sentença condicional if-then-else, da forma:

```
if condição
```

```
then
    comando
else
    comando
fi
```

A condição lógica incluída no "if" costuma consistir numa instrução "test", igual ao caso anterior do "while". As alternativas "then" e "else" referem-se às ações a serem executadas no caso em que a condição lógica comprovada no "if" seja certa ou falsa, respectivamente.

A partícula "fi" indica o final da classe da construção "if".

A seqüência

```
if comando1
then comando 2
fi
```

Pode também ser escrito como

```
comando 1 && comando2
```

Da mesma forma

```
comando 1 || comando2
```

executaria "comando2" somente se "comando1" devolvesse um valor falso. Em cada um dos casos, o valor devolvido pela expressão corresponde ao do último comando simples executado.

O comando man

O exemplo seguinte corresponde ao comando "man" do UNIX, utilizado para imprimir seções do manual do UNIX. A maneira de utilizá-lo é, por exemplo:

```
man sh
man 2 exec
```

No primeiro caso, seria impressa a seção do manual correspondente à shell, começando pela primeira seção ao não ser especificada nenhuma. O segundo exemplo imprimiria a página do

manual correspondente ao comando "exec" na seção 2.

A seguir é mostrada uma versão do comando "man", realizado totalmente na linguagem de comandos da shell:

```
: dois pontos ":" representa um comentário.
: os valores por default são nroff ($N),
: e seção 1 ($s)
N = n s = 1
for i
do case Si in
        [1-9]*) s = $i;;
        —t) N = t;;
        —n) N = n;;
        —) echo opção desconhecida \ $i\ ;;
        *) if test —f man$s/$i.$s
                then S[N] roff man0/S  N   aa manSs/Si.Ss
                else: busca através de todas as seções
                        found = no

                        for j in 1 2 3 4 5 6 7 8 9
                        do if test —f man$j/$i.$j
                                then man $j $i
                                        found = yes
                        done fi
                        case $found in
                            no) echo não existe página $i
                        esac
                fi
        esac
done
```

O programa utilizaria os impressores "troff" ou "nroff", tomando este último por default, ao inicializar N-n. Se entre os argumentos aparecer um número na classe de 1 a 9, este número é tomado como a seção. Após comprovar a existência do comando e a seção especificados, procede a sua impressão, devolvendo uma mensagem em caso contrário.

Substituição de parâmetros

Se um parâmetro da shell não foi iniciado explicitamente, então lhe é atribuído o valor nulo. Por exemplo, se a variável "d" não foi inicializada: /

```
echo $d
```

ou

```
echo ${d}
```

não acontecerá nada, por ser "d" uma cadeia de caracteres nula. Contudo, é possível especificar um valor por default na forma:

```
echo ${d—.}
```

faz com que seja impresso o valor de "d" se esta variável foi inicializada, ou um ponto "." em caso contrário. Analogamente,

```
echo ${d—S1}
```

mostrará o valor de "d" se foi inicializada, ou o valor do parâmetro posicional $1 em caso contrário.

Também é empregada com freqüência a notação

```
echo ${d?mensagem}
```

que mostrará o valor de "d" ou o texto "mensagem", finalizando a execução do programa. Se não foi especificado "mensagem", então obteríamos uma mensagem de erro. Um programa shell que requeira alguns parâmetros poderia então ser inicializada, como:

```
${usuário?} ${acct?} ${bin?}
```

Onde o caracter dois pontos ":" é um comando que não faz nada, uma vez que seus argumentos foram avaliados. Se qualquer das variáveis "usuário", "acct" ou "bin" foram inicializadas, então abandonará a execução do programa.

Substituição de comandos

É possível substituir a saída padrão de um programa de uma forma parecida como atuam os parâmetros. O comando "pwd" ("print working directory", mostra o diretório de trabalho), imprime na saída padrão o nome do diretório atual. Por exemplo, se estivéssemos no diretório "/usr/joao/programas" e se executasse a instrução

```
d = 'pwd'
```

seria equivalente a

```
d = /usr/joao/programas
```

A totalidade da cadeia de caracteres compreendida entre aspas simples (' ') é considerada como um comando a ser executado e o texto é resituado entre (' ') pela saída desse comando.

A substituição de comandos tem lugar em todos aqueles casos onde se realiza a substituição de parâmetros, permitindo o emprego de comandos de processamento de cadeias de caracteres incluídos em instruções da shell. Por exemplo, consideremos o comando "basename" que elimina um sufixo (uma terminação) determinada de uma cadeia de caracteres. Assim:

```
basename main.c.c
```

Produziria como resultado a cadeia "main" ao eliminar a terminação ".c". Isto poderia aparecer como uma parte de uma shell destinada a compilar programas C, na qual se incluiria uma parte como:

```
case $A in
        "...."
        *.c)        B = 'basename $a .c
        "...."
esac
```

inicializando B como a parte correspondente ao nome $A, com a terminação ".c" suprimida.

Ordem de avaliação

A shell é um processador de macros que realiza substituição de parâmetros e comandos, assim como geração de nomes de arquivos. Os comandos são analisados conciliando a seguinte ordem de prioridade e realização de substituições:

- Substituição de parâmetros. Por exemplo $usuário.
- Substituição de comandos. Por exemplo 'pwd'. Somente tem lugar uma avaliação, de maneira que se, por exemplo, o valor da variável X é a cadeia $y, então

```
echo $X
```

mostraria literalmente o texto "$y".

- Interpretação de espaços em branco. De acordo com as substituições anteriores, os caracteres resultantes são partidos em palavras distintas de brancos. Para esta finalidade são considerados "brancos" os caracteres da cadeia $IFS, variável da shell (o mesmo que HOME e PATH, definidas anteriormente). Por default, a cadeia de caracteres $IFS contém um espaço em branco, um tabulador e um salto de linha.
  A cadeia nula não é considerada como uma palavra, a menos que esteja entre apóstrofos ('). Assim:

  ```
  echo ''
  ```

  tomaria um nulo como primeiro argumento do comando echo, enquanto que

  ```
  echo $nulo
  ```

  executará echo sem argumentos se a variável "nulo" não foi iniciada ou seu conteúdo é uma cadeia nula.
- Geração de nomes de arquivos. Explora-se cada palavra buscando os caracteres "*", "?" e [...] e é gerada uma lista alfabética de nomes de arquivos para resituar a palavra. Cada nome de arquivo é um argumento em separado.

Esta lista de substituições que acabamos de descrever, tem lugar para cada um dos argumentos integrantes de um loop "for".

Tratamento de erros

O tratamento dos erros detectados pela shell depende do tipo de erro e se a shell está sendo utilizada ou não interativamente. Uma shell interativa é aquela cuja entrada e saída está dirigida ao terminal do usuário.

A execução de um comando pode produzir um erro por qualquer das seguintes razões:

- Falhas no reendereçamento da Entrada/Saída. Por exemplo, se um arquivo não existe, não pode ser criado ou não pode ser aberto.
- O próprio comando não existe ou não pode ser executado.

- O comando não termina normalmente sua execução devido, por exemplo, a um erro de software ou hardware.
- O comando termina sua execução normalmente porém, devolve um valor diferente de zero em sua finalização.
- Erros de sintaxe. Por exemplo if ... then ... done
- Um sinal software que gera uma interrupção.
- Falha de algum comando agregado, como "cd".

O indicador "—e" da shell causa seu término no caso em que seja detectado algum erro durante a execução.

O quadro seguinte mostra os valores dos sinais software no UNIX:

1		entrada no loop
2		interrupção
3	*	sair
4	*	instrução ilegal
5	*	interrupção por execução "passo a passo"
6	*	instrução IOT ("trap" de Entrada/Saída)
7	*	instrução EMT
8	*	erro na operação em ponto flutuante
9		finalizar (não pode ser detectada ou ignorada)
10	*	erro de bus
11	*	violação de segmentação
12	*	argumento ou chamada ao sistema errôneos
13		escrever em um pipe sem nenhum processamento que o leia
14		sinal de alarme do relógio
15		término software

Os sinais indicados com um "*" produzem uma alteração da memória se não forem invalidados. Os sinais software que possuem interesse para nós são os números 1, 2, 3, 14 e 15.

Os programas em shell normalmente terminam sua execução quando recebem um sinal de interrupção do terminal. Utiliza-se o comando "trap" quando se deseja executar alguma ação antes de finalizar a execução, como eliminar algum arquivo provisório, fechar arquivos, etc... Por exemplo:

```
trap 'rm /tmp/ps$$; exit'2
```

inicia um "trap" (subrotina de tratamento de erros) para o sinal nú-

mero 2 (interrupção), de maneira que quando se recebe o sinal, serão executados os comandos

```
rm /tmp/ps$$; exit
```

"exit" é outro comando da shell que finaliza a execução de um programa. É obrigatório incluí-lo neste caso porque, se não, depois de executar o "trap" a shell prosseguiria a execução à partir da linha na qual se produziu a interrupção.

Os sinais do UNIX podem ser tratados de três maneiras diferentes: podem ser ignorados, em cujo caso os sinais nunca chegarão a ser enviados ao processamento; podem ser detectados, de forma que o processamento decida qual ação tomar quando receber um sinal e, finalmente, podemos deixar que provoquem o término de um processamento sem levar a termo nenhuma ação posterior.

A listagem seguinte mostra o programa "busca", que é um exemplo de um "trap" sem sentença "exit"; explora cada diretório incluído no diretório de trabalho, pergunta seu nome e executa os comandos introduzidos desde o terminal até que alcance o final de arquivo ou receba uma interrupção. As interrupções são ignoradas enquanto que os comandos indicados estejam sendo executados, porém causam o término do programa quando "busca" está esperando uma entrada de dados pela tela.

```
d = 'pwd'
for i in *
do if test —d $d/$i
        then cd $d/$i
                wile echo "$i:"
                        trap exit 2
                        read x
                do trap : 2; eval $x; done
        fi
done
```

"read x" é um comando da shell que aceita uma linha à partir da entrada padrão e deixa o resultado na variável "x". Devolve um valor diferente tanto se alcançar o final do arquivo quanto se for produzida uma interrupção.

Execução de comandos

Para executar um comando diferente dos incorporados pela

mesma, a shell cria um novo processamento por meio da função primitiva "fork". A conjuntura de execução para o comando inclui a entrada e saída padrão e o estado dos sinais, estabelecendo-se no processamento filho antes de executar o comando. Também se utiliza "exec" em alguns casos quando não se deseja realizar um "fork", de forma que simplesmente a shell é resituada por um novo comando. Por exemplo, uma possível versão do comando "nohup" (executa comandos imunes aos sinais do UNIX) seria:

```
trap '' 1 2 3 15
exec $*
```

O "trap" invalida os sinais 1, 2, 3 e 15, que continuarão sendo ignorados pelos comandos que serão executados posteriormente, ao resituar "exec" a shell para cada um dos comandos que figurem no "$*"

CAPÍTULO VI

O NÚCLEO DO SISTEMA UNIX

Este capítulo descreve em termos de alto nível a implementação do núcleo do UNIX. A discussão está dividida em duas partes; a primeira descreve como o sistema UNIX vê os processamentos, usuários e programas, e a segunda descreve o sistema E/S.

O núcleo do UNIX consiste em 10.000 linhas do código C e 1.000 do código Assembler. O código Assembler pode dividir-se em 200 linhas, incluídas por motivos de eficiência (poderiam haver sido escritas em C) e 800 linhas para executar funções hardware que não são possíveis fazer em C. Este código representa de 5 a 10 por cento do que implica a extensa expressão "Sistema Operacional UNIX".

O núcleo é o único código do UNIX que não pode ser substituído por um usuário a seu capricho. Como o núcleo é implementado representa uma grande responsabilidade e um grande poder. Todas as decisões importantes devem ser ponderadas cuidadosamente. Em todas as partes, a simplicidade foi substituída pela eficiência.

Controle de processamentos

No UNIX vários processamentos podem ser executados ao mesmo tempo. Por exemplo, poderíamos executar o programa

sort em "back-ground" e editar outro arquivo em "foreground", enquanto que o sort está sendo executado. Os processamentos que são controlados diretamente pela tela chamam-se processamentos "foreground".

Outros processamentos que foram inicializados, mas que não se possui controle sobre eles, chamam-se processamentos "background". Podemos ter um processamento foreground, porém múltiplos processamentos background podem ser executados simultaneamente. Controlar os processamentos foreground e background é o objetivo desta seção.

Execução e cancelamento de um processamento em background

Para lançar um processamento em background deve-se digitar "&" ao final do comando Ao perder o controle sobre um processamento em background, para anulá-lo não pode utilizar INTERRUPT; deve ser utilizada o comando **kill.**

Para cancelar todos os processamentos do background teclar:

```
$ kill 0
```

Para cancelar somente um processamento, teclar primeiro

```
$ ps
```

O comando **ps** mostra o número de identificações (PIDs) de todos os processamentos ativos para esse terminal.

Por exemplo:

```
$ ps
  PID        TTY        TIME     CMD
3459          03        0:15     —sh
4831          03        1:52     cc program.s
5185          O3        0:00     ps
```

No exemplo acima poderíamos teclar:

```
$ kill 4831
```

onde 4831 é o PID do processamento que queremos cancelar.

O comando ps (estatus do processamento) é empregado para visualizar informação sobre os processamentos ativos. Sua sin-

taxe é:

$ ps [alx] lista-nomes

Opções:

- a — Mostra todos os processamentos associados com um terminal.
- l — Formato estendido
- x — Mostra todos os processamentos não associados com um terminal.

O formato estendido é composto de:

- F — flags associados com os processamentos. Valores de 01.02.04.10 e 20
- S — Estado dos processamentos
  O inexistente
  S processamento dormente (sleeping)
  W esperando
  R executando
  I intermediário
  Z terminado
  T parado
- UID — O ID do usuário do proprietário do processamento, sendo ID o número identificador.
- PID — O número do ID do processamento.
- PPID — O número do ID para os processamentos pais.
- CPU — Utilização dos processamentos para scheduling.
- PRI — Prioridade do processamento, números altos são prioridades baixas.
- NICE — Número usado na computação de prioridade.
- ADDR — Reside na memória, o endereço do núcleo; em outro caso o endereço do disco.
- WCHAN — O sucesso pelo qual o processamento está esperando ou dormindo. Se é um branco, o processamento está em execução.
- TTY — O terminal onde são executados os processamentos.

- TIME — O tempo trancorrido de execução dos processamentos.

Desenvolvimento de pipes no UNIX

Um pipe é um arquivo fictício que um programa pode criar e usar para passar informação à outros programas.

Os pipes são utilizados, principalmente, para passar informação entre programas, como o símbolo da shell ( ), no qual a saída de um programa é a entrada de outro. Isto elimina a necessidade de criar arquivos temporários para passar informação à outros programas.

A biblioteca padrão prevê várias funções de manejo de pipes. As funções **popen** e **pclose** controlam a ambos, um pipe e um processamento. A funçaõ **popen** abre um pipe e cria um novo processamento simultâneo. A funçaõ **pclose** fecha o pipe e espera o término dos processamentos correspondentes. A função **pipe**, por outro lado, dá acesso a baixo nível à um pipe.

A função **popen** cria um novo processamento e abre um pipe ao arquivo padrão de entrada ou saída desse novo processamento. A função tem a forma:

popen (comando, tipo)

Onde **comando** é um ponteiro a uma cadeia de caracteres que contém um comando Shell e **tipo** é um ponteiro a uma cadeia que define se o pipe vai ser aberto para leitura "**r**" ou escrita "**w**".

Esta função normalmente devolverá o ponteiro ao pipe aberto e NULL se encontrar um erro. Vejamos um exemplo:

```
FILE * pstrm;
pstrm = popen ("cat > response", "w");
```

A função **pclose** fecha o pipe aberto pela função **popen**. A função tem a forma:

pclose (apontador)

Onde apontador é um ponteiro ao arquivo do pipe a ser fechado.

Vejamos um exemplo do tema:

```
FILE * pstrm;
pclose (pstrm);
```

A função **pipe** abre um pipe tanto para leitura como para escrita. A função tem a forma:

```
pipe(fd)
```

Onde **fd** é um ponteiro à uma matriz de dois elementos do tipo inteiro. Cada elemento recebe um descritor do arquivo, o primeiro para leitura e o segundo para escrita. A função normalmente devolve 0, e —1 se encontrar algum erro. Vejamos um exemplo:

```
int chan [2]
if (pipe(chan) = = —1)
  exit(2);
```

A função close é empregada para fechar o pipe de leitura ou escrita. Um exemplo é:

```
close (chan [0]) fecha o pipe de leitura
close (chan [1]) fecha o de escrita.
```

APÊNDICE

COMANDOS DO SISTEMA OPERACIONAL

AR — Manejo de arquivos e bibliotecas de arquivos
ar [abclsv] [dmpqrtx] [nome posterior] nome do arquivo arquivo...

a	acrescenta novos arquivos após o nome especificado
b	acrescenta novos arquivos antes do nome especificado
c	suprime a mensagem ao criar o arquivo
l	situa provisoriamente os arquivos no diretório local
v	amplia a informação dada por outras opções
d	elimina dados do arquivo
m	move dados até o final do arquivo
p	lista os nomes dos dados no arquivo
r	substitui dados no arquivo
s	força a regeneração da tabela de símbolos do arquivo
t	lista a tabela de conteúdos dos arquivos
x	extrai dados do arquivo
nome posterior	posições do nome dos arquivos; usualmente empregado com **a**, **b**, **i**, **r** e **m**
nome do arquivo	nome do arquivo
arquivo...	um ou mais dados a serem colocados no arquivo

AS — Assembler
as [-o objfile] [-n] [-m] [-R] [-r] [-v] nome do arquivo

—n	põe/tira endereços compridos/curtos

—m	executa o pré-processador de macros m4 à entrada do Assembler
—R	elimina o arquivo de entrada ao finalizar o Assembler
—r	põe todos os dados Assembler na seção .text
—v	escreve o número da versão do Assembler no arquivo de erros padrão
nome	nome do arquivo Assemblar

AT — executa comandos na data e hora especificadas

at hora [dia] [arquivo]

hora [APNM]	especificada a hora e minuto. **A** indica **AM**, **P** indica **PM**, **N** indica tarde e **M** indica meia noite.
dia	refere-se tanto a um nome do mês seguido por um número do dia, ou, opcionalmente, um dia da semana
arquivo	arquivo a ser empregado posteriormente como entrada para a **shell**.

AWK — reconhecimento de modelos e linguagem de processamento

awk [-Fc] [modelo ação] [arquivo]

-Fc	emprega o caracter **c** como separador de campos
modelo	conjunto de modelos a reconhecer
ação	ação a realizar quando o modelo é encontrado
arquivo	um ou mais arquivos a buscar

BASENAME, DIRNAME — isola partes de nomes de arquivos

basename	cadeia de caracteres [sufixo]
dirname	cadeia
cadeia	indica cadeia de caracteres a buscar
sufixo	indica sufixo (terminação) a eliminar do nome

BC — linguagem aritmética de precisão arbitrária

bc [-c] [—l] [arquivo...]

—c	somente compila
—l	carrega a biblioteca matemática de precisão arbitrária
arquivo	um ou mais nomes de arquivos

BDIFF — compara dois arquivos e informa suas diferenças

bdiff arquivo1 arquivo2 **[—n] [—s]**

arquivo1	primeiro nome do arquivo
arquivo2	segundo nome do arquivo
—n	compara um número (n) de linhas
—s	suprime diagnósticos

CAL — imprime calendário

cal [mês] ano

mês	número de dois dígitos correspondente ao mês
ano	número de quatro dígitos correspondente ao ano

CAT — concatena e imprime arquivos

cat [—e] [—n] [—s] [—t] [—u] [—v] arquivo...

—e	coloca o caracter **$** ao final da linha quando é empregado com a opção **—v**
—s	omite mensagens de erro sobre arquivos inexistentes
—t	coloca o caracter **I** como tabulador quando é empregado conjuntamente com a opção **—v**
—u	causa que a saída seja sem buffer (caracter por caracter)
—v	faz com que sejam mostrados os caracteres não imprimíveis
arquivo	um ou mais nomes de arquivos

CB — copia um programa C colocando espaçamento e identificação

cb [—s] [—j] [—l comprimento] [arquivo]

—s	converte o código ao estilo proposto na **Linguagem de programação C** por **Kernighan e Ritchie**
—j	reúne as linhas partidas
—l comprimento	comprimento, parte as linhas maiores que o comprimento
arquivo	um ou mais arquivos de programas em C para reformatar

CC — Compilador de C.

cc [—c] [—Bserie] **[—E] [—O(KS] [—**O volume] **[—P] [—p] [—S] [—t[p012a1]] [—v] [—voc,**arg1 [,arg2..]] arquivo...

—Bserie	constrói nomes de arquivos para

	substituir as fases do pré-processador, compilador, Assembler e link-editor
—c	suprime a fase de link-edição
—E	executa somente **cpp** sobre os programas C indicados
—O(KS)	otimiza o código objeto **K** — otimizações dependentes do kernel (núcleo) **S** — otimiza o emprego do stack (pilha)
—o saída	nome do módulo de saída link-editado
—P	executa somente o pré-processador, com saída sobre o arquivo de sufixo "**.i**"
—p	monitoriza a execução
—S	produz a saída em linguagem Assembler sobre um arquivo com sufixo "**.s**"
—t[p012a1]	utiliza somente o pré-processador, compilador, Assembler e link-editor correspondentes aos nomes de arquivos construídos com a opção — **B.**
—v	mostra o nome de cada sub-processamento
—wc,arg1[,arg2...]	elimina os argumentos de passagem **c** pertencentes ao conjunto **p012a1**
arquivo	um ou mais nomes de arquivos fonte em linguagem C

CD — muda o diretório de trabalho atual

cd [diretório]

diretório	nome do novo diretório de trabalho; por default é o valor de **$HOME**

CHMOD — troca a forma de acesso a determinados arquivos

chmod [absmode] [symmode] arquivos

abmode	4000 — inicializa o número do usuário em execução. 2000 — inicializa o número do grupo em execução.

1000 — guarda o texto depois da execução.
0400 — permite leitura pelo proprietário.
0200 — permite escrita pelo proprietário.
0100 — permite execução pelo proprietário.
0070 — leitura, escrita e execução por grupo.
0007 — leitura, escrita e execução pelos usuários restantes.

symmode [quem] op permissão [op permissão]

quem
u utiliza
g grupo
o outro

op
+ acrescenta permissão à forma de acesso ao arquivo
tira permissão
= atribui permissão absoluta

permissões
r leitura
w escrita
x execução
s inicializa número identificador do proprietário ou grupo
t guarda o texto

arquivos arquivo(s) a mudar

CHOWN, CHGRP — muda o número identificador do proprietário ou grupo

chown proprietário do arquivo...
chgrp grupo do arquivo...
proprietário indica o número decimal do usuário ou o nome do **login.**
grupo indica o número decimal do grupo ou um nome do grupo correspondente ao arquivo de grupos.
arquivo um ou mais arquivos a mudar o proprietário ou grupo especificados.

CLEAR — elimina a tela

clear

CMP — compara dois arquivos e informa suas diferenças

cmp [—l] [—s] arquivo1 arquivo2

—l	mostra o número de bytes que diferem
—s	não mostra as diferenças
arquivo1	nome do arquivo
arquivo2	nome do arquivo

COL — elimina os line-feeds (saltos de linha) inversos

col [—bfpx]

—b	suprime a possibilidade de backspace
—f	suprime os saltos de meia linha
—p	mostra as seqüências de escape desconhecidas encontradas na entrada como caracteres normais, podendo ocorrer sobre-impressão devido aos saltos de linha inversos
—x	suprime a conversão de espaços em branco em tabuladores

COMM — mostra as linhas comuns a dois arquivos classificados

comm	[—[123]] arquivo1 arquivo2
—1	suprime linhas somente do arquivo1
—2	suprime linhas somente do arquivo2
—3	suprime linhas de ambos os arquivos
arquivo1	nome do primeiro arquivo
arquivo2	nome do segundo arquivo

CP — copia o conteúdo de um arquivo em outro arquivo ou diretório

cp	arquivo ... objeto **arquivo** ... um ou mais nomes de arquivos antigos **objeto** ... novo nome do arquivo
cp	arquivo ... dire**tório** **arquivo** ... um ou mais **nomes** de arquivos **diretório** ... nome do diretório

CRYPT — codifica/decodifica arquivos

crypt [palavra chave]

palavra chave	chave de conversão

CU — chama outro sistema UNIX, um terminal ou um sistema não UNIX

cu [—llinha] [—svelocidade] [—t] [—h] [—d] [—m] [—o!—e] telno /dir

—llinha	especifica o nome do arquivo para a li-

	nha de comunicação do dispositivo; por default é **/dev/ttya**
—svelocidade	especifica a velocidade de transmissão; velocidade pode ser 50, 75, 110, 134, 150, 300, 1200, 1800, 2400, 4800 ou 9600; por default é 300 bauds
—t	indica que a chamada é a outro terminal
—h	emula eco local, suportando chamadas a outros sistemas que requeiram terminais em modo half-duplex
—d	realiza execução passo a passo para diagnósticos
—o	gera paridade ímpar para os dados enviados ao terminal remoto
—e	gera paridade par para os dados enviados ao terminal remoto
telno	número de telefone da linha
dir	indica linhas de conexão direta

DATE — mostra e inicializa a data e hora do sistema

date	[+ formato] [mmddhhmm[.ss] [YY]]
YY	**dois** dígitos do ano
mm	**dois** dígitos do mês
dd	**dois** dígitos do dia
hh	**dois** dígitos da hora do dia
mm	**dois** dígitos do minuto da hora
ss	dois dígitos do segundo do minuto
+ formato	formato de saída controlável pelo usuário

n	insere um caracter de salto de linha
c	insere um caracter de tabulação
D	data em formato mm/dd/yy
H	hora: de 00 a 23
M	minuto: de 00 a 59
S	segundo: de 00 a 59
T	hora em formato HH:MM:SS
j	dia do ano: 001 a 366
w	dia da semana · domingo = 0
a	dia da semana abreviado - domingo a sábado
n	mês abreviado janeiro a dezembro
r	hora em forma AM/PM

DD — realiza conversões e copia arquivos

dd [opção = valores]...

if = ifile	especifica **ifile** como arquivo de entrada; por default é a entrada padrão
of = ofile	especifica **ofile** como arquivo de saída; por default é a saída padrão
ibs = n	muda o tamanho do bloco de entrada a "n" bytes; por default é 512.
obs = n	muda o tamanho do bloco de saída a "n" bytes; por default é 512
bs = n	muda o tamanho do bloco de entrada e saída a "n" bytes; por default é 512
cbs = n	muda o tamanho do buffer de conversão a "n" bytes
skip = n	salta "n" registros de entrada antes de começar a cópia
seek = n	salta "n" registros de saída antes de começar a cópia
count = n	copia somente "n" registros de entrada
conv = tipo	onde tipo é um dos seguintes: **ascii** converte de EBCDIC a ASCII **ebedic** converte de ASCII a EBCDIC **ibm** outra conversão de ASCII a EBCDIC **lcase** converte letras a minúsculas **ucase** converte letras a maiúsculas **swab** intercambia pares de bytes **noerror** não interrompe o processamento se houver erro **sync** reenche cada registro de entrada ao número de caracteres especificado por **ibs** (bloqueio de entrada) "...,..." separa os tipos de conversão através de aspas

DIFF — mostra as diferenças entre dois arquivos

diff [—efbn] arquivo1 arquivo2

—e	gera um arquivo de comandos do editor para fazer arquivo2 a partir do arquivo1
—f	produz um texto similar ao **—e** não utilizável com o editor, em ordem inversa
—b	ignora os brancos posteriores
—h	produz diferenças mais rapidamente

arquivo1	nome do primeiro arquivo
arquivo2	nome do segundo arquivo

DU — mostra a quantidade de disco utilizado

du [—a] [—r] [—s] [nome...]

—a	proporciona uma entrada por arquivo
—r	fornece mensagens acerca dos diretórios que não podem ser lidos, abertos, etc.
—s	mostra somente as somas totais
nome	nome do diretório ou arquivo

ECHO — mostra argumentos na tela (para a shell)

echo	[arg]...
arg	informação a mostrar no terminal

ED, RED editor de textos

ed [—] [—x]	[arquivo]
red [—] [—x]	[arquivo]
—	suprime caracteres contados por **e**, **r** e **w**
x	maneja arquivos lacrados
arquivo	nome do arquivo a ser lido no buffer do editor

O seguinte é um sumário dos comandos do editor em ordem alfabética. O número da linha por default é especificada como (.) a menos que se indique outra coisa.

(.)a	o comando **acrescentar**, acrescenta o texto de entrada ao buffer depois da linha especificada. Introduzir um ponto (.) como primeiro e único caracter termina a inserção;
(.)c	o comando **change** muda as linhas especificadas e aceita o texto de entrada que as substitui. Introduzir um ponto (.) como primeiro e único caracter para terminar as mudanças;
(.,.)d	o comando **delete** troca a linha ou classe de linhas especificadas desde o buffer;
e nome do arquivo	o comando **edit** elimina o conteúdo do buffer e copia o arquivo especifi-

	cado no buffer;
E nome do arquivo	o comando **edit** troca o conteúdo do buffer e copia o arquivo especificado no buffer;
f nome do arquivo	o comando **file** dá novo nome ao arquivo em uso com o nome do arquivo especificado;
(1,$)g/re/command	comando **global**, interativo marca cada linha na qual aparece a máscara /re/ e aceita o comando em cada linha marcada;
(1,$)G/re/	o comando global interativo marca cada linha na qual aparece a máscara /re/ e aceita o comando em cada linha marcada.
h	o comando **help** dá uma explicação do diagnóstico "**?**" mais recente;
H	o comando **help** dá uma explicação do anterior e subseguintes diagnósticos "**?**";
(.)i	o comando **insert** insere o texto de entrada no buffer depois da linha especificada. Introduzir um ponto (.) como primeiro e único caracter para terminar a inserção
(.,. + 1)j	o comando **join** junta linhas contíguas eliminando o caracter de salto de linha;
(.)kx	o comando **mark** marca a linha especificada com o nome "x";
(.,.)l	o comando **inst** imprime as linhas direcionadas e assinala caracteres não imprimíveis;
(.,.)ma	o comando **move**, move a linha especificada depois da linha "a";
(.,.)n	o comando **number** lista o arquivo assinalando os números da linha;
(.,.)p	o comando **print** lista as linhas do texto especificadas;
P	o comando **prompt** solicita com um asterisco (*) os comandos seguintes;

q	o comando **quit** finaliza a sessão de edição e comprova se foram feitas as mudanças no buffer desde a última execução do comando w;
Q	o comando **quit** finaliza a sessão de edição e não verifica se foram produzidas as mudanças no buffer desde a última execução do comando w;
($)R arquivo	o comando **read** realiza uma cópia do arquivo depois da linha especificada;
(...)s/re/rel/	o comando **substituye**, substitui a primeira ocorrência em uma linha da máscara (re) pela máscara (rel).
(...)s/re/rel/g	o comando **substituye** realiza uma troca total da máscara (re) pela máscara (rel) nas linhas especificadas;
(...)ta	o comando **copy** copia as linhas especificadas depois da linha "a";
u	o comando **undo** anula os efeitos do comando mais recente que tenha modificado o buffer;
(...)v/re/rel	substitui globalmente linhas nas quais não tenha sido especificada a máscara (re);
(...)V/re/	o comando **global** interativo marca cada linha na qual não se tenha especificado a máscara (re) e aceita um comando para cada linha marcada;
(1,S)w nome arquivo	o comando **write** escreve o conteúdo do buffer no arquivo que tenha sido alterado o nome;
x	lacra o arquivo;
($) =	número da linha mostrada na tela;
!comando shell	o restante da linha a partir do caracter "!" é enviada ao sistema operacional interpretando-se como um comando;
(. + 1)〈nova linha〉	comando de impressão que mostra linha endereçada.

ENABLE, DISABLE — permite/impede a utilização de uma impressora

enable impressoras
disable [—C] [—R[razão] impressoras
—c
—r[razão] associa uma razão com a desativação das impressoras;
impressoras uma ou mais impressoras a permitir/impedir seu emprego.

ENV — Inicializa o conjunto de execução de um usuário (environment);
env [—] [nome = valor] ... [comando args]

EXP — Avalia expressões como argumentos para a shell;
exp arg ...
arg um ou mais argumentos a serem avaliados
exp expressões e argumentos com a ordem de prioridade;

x1 /! x2	devolve x1 se for diferente de nulo ou 0, se não devolve x2;
x1 /& x2	devolve x1 se x1 ou x2 (são ambos ao mesmo tempo) são diferentes de nulo ou 0, se não devolve 0.
x1/ < x2	compara se x1 é menor que x2, devolvendo 1 se for certo e 0 se for falso;
x1 / < = x2	compara se x1 é menor ou igual a x2 devolvendo 1 se for certo e 0 se for falso;
x1 = x2	compara se x1 e x2 são iguais. Devolve 1 se for certo e 0 se for falso;
x1 != x2	compara se x1 é diferente de x2. Devolve 1 se for certo, 0 se for falso;
x1 / x2	compara se x1 é maior que x2. Devolve 1 se for certo e 0 se for falso;
x1 < =x2	compara se x1 é menor ou igual a x2. Devolve x1 se for certo e 0 se for falso;
x1 + x2	soma x1 a x2;
x1 - x2	subtrai x2 de x1;
x1 * x2	multiplica x1 por x2;
x1 / x2	divide x1 por x2;
x1 % x2	devolve o valor de x1 restante módulo x2.

FILE — determina o tipo de um arquivo.
arquivo [—c] [—f ffile] **[—m** mfile] arquivos

—f ffile	arquivo contendo os nomes de dados a examinar;
—m mfile	utiliza **mfile** como "número mágico" (indicador do tipo de arquivo);
—c	comprova o "número mágico" para detectar erros de formato;
arquivos	um ou mais arquivos a examinar.

FIND — Busca arquivos.

find pathname - list - expressão.

pathname-list	um ou mais nomes de arquivos;
expressão	busca o modelo indicado, onde **n** é um inteiro, **+n** é maior do que **n** e **—n** é menor do que **n**.

Expressões permitidas:

—atime n	certo se o arquivo houver sido acessado nos últimos **n** dias;
—ctime n	certo se o arquivo houver sido mudado nos últimos **n** dias;
—exec comando	certo se a execução do **comando** devolve um 0;
(expressão)	certo se a expressão entre parênteses "( )" for certa;
—group gname	certo se o arquivo pertence ao grupo **gname**;
—links n	certo se o arquivo possui "n" enlaces (links);
—mtine n	certo se o arquivo foi modificado em "**n**" dias;
—name nome arquivo	certo se o nome do arquivo coincide com o dado;
—ok comando	o mesmo que o comando **exec**, porém é executado somente após receber confirmação por parte do usuário;
—newer nome arquivo	certo se o nome do arquivo foi modificado antes que os dados;
—perm onum	certo se coincidirem os indicadores da forma de acesso em octal;
—print	mostra o nome completo do arquivo;
—size n	certo se o arquivo ocupa "**n**" blocos

de comprimento;

—type x	certo se o arquivo tiver o tipo "**x**", onde "**x**" pode ser um "**b**" para um arquivo especial organizado em forma de blocos, "**c**" para um arquivo especial organizado em forma caracter, "**d**" para um diretório e "**f**" para um arquivo normal;
user nome	certo se o arquivo pertencer ao usuário.

GREP, EGREP, FGREP — Busca de cadeias e expressões.

grep [—v] [—c] [—l] [—n] [—b] [—s] limexpr [arquivos];
egrep [—v] [—c] [—l] [—n] [—b] [—e expressão] [—f arquivos] [regexpr] [arquivos];
fgrep [—v] [—x] [—c] [—l] [—n] [—b] [—e expressão] [—f arquivos] [strings] [arquivos]

—v	imprime todas as linhas exceto as que são iguais;
—x	imprime somente as linhas que são completamente iguais (fgrep);
—c	imprime um contador das linhas iguais;
—l	lista nomes de arquivos com linhas iguais;
—n	cada linha precedida pelo número da linha;
—b	cada linha precedida pelo número do bloco que foi construído;
—s	imprime mensagens de erro para arquivos inexistentes (grep);
—e expressão	premite às expressões começar com — (egrep, fgrep);
—f arquivo	colhe a expressão regular (egrep) ou lista de strings (fgrep) do arquivo;
limexpr	busca a expressão especificada; deve limitar-se a expressões regulares da forma do ed;
regexpr	busca uma expressão regular;
strings	busca um string;
arquivos	para buscar em um ou mais arquivos.

HEAD — mostra as primeiras linhas de um arquivo

head [ɔntador] [arquivo...]

—contador	proporciona um número especificado de linhas;
arquivo...	um ou mais nomes de arquivos.

KILL — finaliza um processamento

kill [—sig] identificador-processamento...

—sig	1 — entrada em loop (SIGNUP); 2 — interrupção (SIGINT); 3 — sair (SIGQUIT); 4 — instrução ilegal (SIGILL); 5 — trace trap (SIGTRAP); 6 — instrução IOT (SIGIOT); 7 — instrução EMT (SIGEMT); 8 — erro na operação em ponto flutuantes (SIGFPE); 9 — finalizar (SIGBUS); 10 — erro do bus (SIGBUS); 11 — violação de segmentação da memória (SIGSEGV); 12 — erro na chamada ao núcleo (system call) (SIGSYS); 13 — escrever num pipe sem nenhum processamento em leitura (SIGPIP); 14 — alarme por temporização (SIGALARM); 15 — terminação software (SIGTERM); 16 — não atribuído.
id. de processamento	um ou mais números de identificação de processamentos.

LINT — verificação de programas em C

lint [—abhlnpuvx] arquivo...

—a	suprime erros de atribuições de valores long à variáveis inteiras;
—b	suprime erros de sentenças não alcançadas;

—h	não tenta encontrar erros de programação, nem estilo não apropriado e reduz comentários;
—lx	inclui a biblioteca lint llib-lix.ln;
—n	não verifica compatibilidade com a biblioteca padrão;
—p	verifica portatibilidade ao IBM e GCOS;
—u	suprime mensagens de funções e variáveis sem utilizar;
—v	suprime mensagens de argumentos de funções sem utilizar;
—x	suprime mensagens de variáveis externas nunca utilizadas;
arquivo...	um ou mais nomes de arquivos.

LOGIN — conexão ao sistema

login [nome do usuário] [env-var ...]].

nome do usuário	muda um usuário já conectado ao sistema por outro usuário;
env-var	argumentos para expandir ou modificar o conjunto de execução (environment).

LP — gestiona o manejo de uma impressora

lp [—c] [—d dest] [—m] [—n número] [—o opção] [—s] [—t título] [—w] arquivos.

—c	faz cópias dos arquivos a imprimir imediatamente antes da chamada;
—d dest	envia petição ao dest;
—m	envia mensagem depois de impressos os arquivos;
—n número	número de cópias; por default é 1;
—o opção	especifica as opções que dependem da impressora;
—s	suprime as mensagens do lp;
—t título	imprime o título no início da página;
—w	escreve uma mensagem no terminal depois de imprimir os arquivos;
arquivos	um ou mais arquivos para serem impressos.

LS — lista o conteúdo de um diretório

ls [—AaCcdFfgiLlqRrstul].

—A	lista todas as entradas exceto "." e "..";
—a	lista todos os arquivos;
—C	força a saída em multi-colunas;
—c	usa a data de criação do arquivo para sua classificação (—t) ou impressão (—l);
—d	somente lista o nome se o argumento for um diretório;
—F	marca os diretórios com uma barra "/", os arquivos executáveis com um "*", os pipes com " = " e os links simbólicos (enlaces) com "@";
—f	faz com que cada argumento seja interpretado como um diretório;
—g	inclui o número identificador do grupo em formato comprido;
—i	escreve o número de i-no na primeira coluna de cada arquivo;
—L	lista o arquivo ou enlaces de referências a diretórios se seu argumento for um link (enlace) simbólico;
—l	lista em formato expandido;
—q	faz a impressão dos caracteres não gráficos nos nomes do arquivo;
—R	lista sub-diretórios;
—r	classifica em ordem inversa;
—s	fornece o tamanho em blocos;
—t	ordena pela data da modificação;
—u	classifica pela data do último acesso;
—1	faz um arquivo por linha.
nome...	um ou mais nomes de diretórios de arquivos.

M4 — processador de macros em texto

m4 [—Bint] [—e] [Hint] [—Sint] [—s] [—Tint] [—D nome] [= val1] [—U nome] [arquivos]

— Bint	Muda o tamanho da pilha e buffers de argumentos; por default 4096 bytes;
—e	opera interativamente;
—Hint	muda o tamanho da tabela de símbolos; por default 199 bytes;
—Sint	Muda o tamanho do stack; por default 100 slots;
—s	permite linhas sync

—Tint	muda o tamanho do buffer de comandos; por default 512 bytes;
—D nome	[= val] define o nome como "val" ou o valor nulo se não for especificado "val";
—U nome	não define nomes;
arquivos	um ou mais nomes de arquivos a processar.

MAIL, RMAIL, SMAIL — envia/lê mensagens a/de usuários

mail [—t] usuários
mail [—f] arquivos [—epqr]
rmail [—t] usuários
smail [—t] usuários

usuários...	usuário(s) a quem é enviada a mensagem
—f	arquivos envia mensagem à arquivo em lugar de arquivos mail (default);
—e	não imprime mensagens, devolve 0 se o usuário tiver mensagens ou 1 se não as tem;
—p	imprime o arquivo de mensagens;
—q	produz o término do mail após uma interrupção,
—r	ordem "primeiro a entrar primeiro a sair" (FIFO);
—t	coloca no "tampão" os nomes de todas as pessoas às quais foi enviada a mensagem.

Comandos interativos:

d	elimina mensagens e continua com a seguinte
EOT (control-D)	fecha o arquivo de mensagens e termina
m [usuários]...	envia a mensagem a um ou mais usuários
tecla newline	continua com a mensagem seguinte
P	imprime a mensagem novamente
q	fecha o arquivo de mensagens e termina
s [arquivo]	salva a mensagem no arquivo nomeado
X	sai sem mudar o arquivo de mensagens
w [arquivo]	salva a mensagem no arquivo nomeado sem cabeçalhos

!comando	executa um comando da shell
?	imprime o resumo dos comandos
—	volta a mensagem anterior

MKDIR — **cria um diretório**

mkdir dirname

dirname	um ou mais nomes de diretórios

MORE, PAGE — **filtro de saída da tela para realizar paginação de arquivos**

More [—cdfslu] [—n] [+ número de linha! + /pattern] page

[nome...] [—cdfslu]

—c	elimina cada linha antes de sobreimprimí-la
—d	mostra a mensagem "pressionar espaço para continuar" ("Hit return to continue")
—f	não trunca as linhas de comprimento excessivo
—s	reduz múltiplas linhas em branco a um só
—l	não trata os ^ L (form feed) de forma especial
—u	não permite o sublinhado no terminal
—n	especifica o número das linhas a representar
+ número de linha	começa pelo número indicado
+ /pattern	começa duas linhas depois da que contém a máscara
nome...	um ou mais arquivos a visualizar

Comandos interativos:

[i] 〈espaço〉	mostra "i" linhas adicionais ou outra tela se não for especificado o argumento
ôD	mostra 11 linhas mais
v	começa o editor vi na linha em curso
h	ajuda
'	volta ao ponto no qual se iniciou a última busca
i/expr	busca a ocorrência iésima da expressão

	(expr)
if	salta "i" telas
is	salta "i" linhas
iz	especifica uma nova janela de tamanho "i"
q	sai do comando more
!cmd	chama o comando shell
:f	mostra o nome e o número de linha do arquivo em curso
	repete o comando anterior

NICE — modifica a prioridade de processamento

nice [—número] comando [argumentos]

—número	número de prioridade; a maior prioridade é 0 e a menor 19
comando	comando a executar com a prioridade mais baixa
argumentos	argumentos para o comando especificado

NROFF — formata arquivos de texto

nroff [—c nome] [—e] [—h] [—i] [—k nome] [—m nome] [—nN] [—o lista] [—q] [—raN] [—sN] [—T nome] [un] [—z] [arquivo]

—c nome	utiliza arquivos de macros usr/lib/macros/cmp.[nt].[dt].nome e usr/lib/macros/ucmp.[nt].nome
—e	separa as palavras ajustando as linhas
—h	usa os tabuladores horizontais
—i	lê as entradas padrão depois dos arquivos
—k nome	coloca os arquivos de macros computados em [dt].nome
—m nome	prepara o arquivo de macros /usr/lib/tmac/tmac.nome
—nN	usa N como primeira página
—o lista	lista as páginas especificadas na saída ou separadas por uma classe (N-M)
—q	realiza entrada/saída simultânea ante um comando "rd"
—raN	inicializa o registro a N
—sN	detém a listagem a cada N páginas

—T nome	envia à saída o nome do terminal especificado
—un	inicializa o número de sobre-impressões (em negrito) a "n"
—z	imprime somente as mensagens geradas pela petição ".tm"
arquivo	arquivo para ser formatado

OD — volta em octal

od [—bcdosx] [arquivo] [[+] offset [.] [b]]

—b	interpreta os bytes em octal
—c	interpreta os bytes em ASCII
—d	interpreta palavras em decimal sem sinal
—o	interpreta palavras em octal
—s	interpreta palavras de 16 bits em decimal com sinal
—x	interpreta palavras em hexadecimal
arquivo	arquivo a voltar
+	especifica se foi omitido o nome do arquivo anterior
offset	especifica o deslocamento (offset) inicial
.	interpreta o deslocamento em decimal
b	interpreta o deslocamento em blocos de 512 bytes

PASSWD — muda a chave de acesso

passwd	[nome]
nome	nome do usuário

PR — imprime arquivos

pr [—a] [—d] [—eck] [—f] [—h] [—iick] [—ln] [—m] [—n] [+n] [—nck] [—ok] [—p] [—r] [—sc] [—t] [—wn] arquivo...

—a	imprime em várias colunas na largura da página
—d	saída com duplo-espaço
—eck	expande os tabuladores em sua entrada como espaços em branco "c" se retiver o caracter de tabulação de saída
—f	lista com saltos de linha
—h hd	usa hd como cabeçalho em lugar do nome do arquivo

—ick	substitui os espaços em branco por tabuladores
—ln	usa "n" como comprimento da página, por default é 66
—m	imprime todos os arquivos em colunas separadas
—n	saída em "n" colunas
+n	começa na página "n"
—nck	utiliza uma largura de "k" dígitos para numeração de linhas (por default "k" é 5)
—ok	desloca cada linha em "k" caracteres
—p	realiza uma pausa antes de imprimir cada página
—r	não imprime erros de abertura de arquivos
—sc	separa as colunas com o caracter "c"
—t	não imprime as 5 linhas de cabeçalho ou rodapé da página
—wn	largura de página de "n" caracteres, por default é 72
arquivo	um ou mais nomes de arquivos

PRINT/LPR — envia arquivos ao spooler para imprimí-los

print ! lpr [—b] [—cp nnn] [—fm xxxxx] [—ln nn] [—pr nn] [—pt lpnn] arquivos

—b	imprime com cabeçalho (banner)
—cp nnn	especifica o número de cópias (nnn) a imprimir, por default 1
—fm xxxxx	designa a cola de impressão correspondente ao tipo de formulário (xxxxx)
—ln nn	especifica o número de linhas por polegada (nn), por default 6
—pr nn	especifica o número de prioridade (nn) para a impressão
—pt lpnn	reencaminha a saída à impressora especificada (lpnn)
arquivos	um ou mais nomes de arquivos a imprimir

PRINTENV — imprime as variáveis do conjunto de execução (environment)

printenv

PS — lista informação sobre processamentos ativos

ps [—a] [—d] [—e] [—f] [—g GLIST] [—l] [—n nome da listagem] [—p plist] [—s swapdev] [—t tlist] [—u ulist]

—a	lista informação sobre todos os processamentos associados com o terminal
—d	lista a informação de todos os processamentos, exceto os processamentos do grupo
—e	lista informação de todos os processamentos
—f	gera uma listagem completa
—g glist	lista somente os processamentos do grupo especificados no glist
—l	gera uma listagem em formato expandido
—n lista de nomes	usa lista de nomes como alternativa
—p plist	lista somente os processamentos especificados
—s swapdev	usa swapdev em lugar de /dev/swap/
—t tlist	lista somente os processamentos associados aos terminais especificados em tlist
—u ulist	lista somente os processamentos associados aos usuários especificados em ulist.

RM, RMDIR — elimina arquivos ou diretórios

rm [—f] [—i] [—r] arquivo...
rmdir arquivo...

—f	faz separações
—i	pergunta interativa antes de eliminá-los
—r	elimina um diretório e todos os arquivos do mesmo
arquivo	um ou mais nomes de arquivos ou diretórios

SH — linguagem de programação de comandos

sh [—ceiknrstuvx] [argumento].....

—c string	usa string como um comando

—e	um mal status de saída se não interativo
—i	entra em forma interativa
—k	coloca palavras chave em um entorno para um comando
—n	lê, mas não executa comandos
—r	restringe a conjuntura
—s	lê desde a entrada padrão
—t	sai depois de executar um comando
—u	trata as variáveis não agrupadas como um erro
—v	imprime as linhas de entrada quando são lidas
—x	imprime o comando quando executa
arg...	um ou vários argumentos

Caracteres especiais da shell

;	executa seqüencialmente o procedimento pipeline
@	executa sem esperar o término do comando
@	executa o comando se o comando anterior devolveu 0
'...'	cita todos os caracteres que estão limitados entre apóstrofos
/	cita todos os caracteres que se encontram depois da barra
"..."	comandos e parâmetros a serem substituídos
# text	considera todo o texto introduzido até o final da linha como comentário

Geração de um nome de arquivo da shell

*	qualquer string de caracteres
?	qualquer caracter
[..]	quaisquer dos caracteres contidos

Sentenças de comandos da shell

case palavra in [pattern]
[... list;;] ...esac
for name [in palavra..] do lista done
if lista then lista [elif lista then lista]...[else lista] fi
(lista)
{lista}

while lista [do lista] done

Parâmetros de substituição da shell

$ #	número de parâmetros
$—	flag fornecido por chamada
$?	valor retornado do último comando executado
$$	número de processamento da shell
$!	número do último comando (background)
$HOME	especifica o diretório home
$IFS	especifica separadores de campo interno
$MAIL	especifica o arquivo mail
$PATH	especifica o arquivo PATH
$CDPATH	especifica a busca
$PS1	especifica o prompt primário, $ por default
$PS2	especifica o prompt secundário, > por default
$TERM	especifica tipo de terminal
nome = valor	especifica valores de parâmetros
$ { parâmetro }	utiliza um conjunto de parâmetros
$ { parâmetro:? palavra }	utiliza parâmetros se estiver agrupado e for nulo, se não imprime palavra e depois sai
$ { parâmetro: + palavra }	utiliza palavra se o parâmetro estiver agrupado ou for nulo, em outro caso nada
< palavra	a palavra do arquivo como entrada padrão
<< palavra[—]	utiliza a palavra do arquivo como entrada padrão seguido pela identificação da linha, palavra ou fim do arquivo
> palavra	utiliza a palavra do arquivo como saída padrão
>> palavra	utiliza a palavra do arquivo como saída padrão, acrescentan-

	do-o à saída se o arquivo já existe
>fdígito	duplica a entrada padrão para o dígito do descritor do arquivo
<f—	fecha a entrada padrão

Comandos especiais da SH

.arquivo	lê e executa comando do arquivo e volta
break [n]	sai de um for ou while em "n" níveis
cd[arg]	troca o diretório atual a arg; diretório home por default
continue [n]	resume n-interação de um for ou while
eval [arg.]	lê e executa argumentos
exec [arg.]	executa argumentos em lugar da shell
exist [n]	sai com o status "n"
export [name.]	exporta o nome para um conjunto de comandos
login [argl]	muda de usuário
newg.[arg...]	muda de grupo
read name...	lê da entrada padrão
readonly [nome...]	marca o nome como somente leitura

set [ekntuvx arg...]]

—e	status mal se não interativo
—k	coloca palavra chave em um conjunto para um comando
—n	lê porém não executa comandos
—t	sai depois de executado um comando
—u	trata as variáveis não agrupadas como um erro
—v	imprime linhas de entrada quando são lidas
—x	imprime comandos quando o executa
——	não muda qualquer das opções
arg...	um ou mais argumentos

shift	dá novo nome aos argumentos
test	avalia expressões condicionais
times	imprime tempo de processamentos acumulados

trap [arg][n]

—f	impõe como tamanho limite "n" blocos nos arquivos dos processamentos filhos
—p	muda o tamanho do pipe a "n"

unmask [nnn] agrupa a máscara do arquivo ao valor nnn

wait [n]	espera pelo número [n] de identificação especificado e imprime o status do término
file	lê e executa comandos do arquivo

SIZE — lista o tamanho de um arquivo objeto

size [—x] [—o] [—v] [objeto...]

—x	imprime números em hexadecimal
—o	imprime números em octal
—v	imprime a informação da versão
objeto	um ou vários nomes de arquivos objetos

SLEEP — suspende a execução por um intervalo de tempo

sleep time

time	suspende pelo número especificado em segundos

SORT — ordena ou mescla arquivos

sort [—cmu] [—o nome] [—dfinr] [—btx] [+ posição] [—posição 2] [arquivos]

—b	ignora brancos repetitivos
—c	verifica se os arquivos de entrada estão classificados de acordo com as regras
—d	classifica em ordem de dicionário
—f	igual tratamento maiúsculas e minúsculas
—i	ignora caracteres fora do código ASCII desde o 40 até o 176
—m	mescla arquivos (merge)
—n	classifica em ordem aritmética
—r	classifica em ordem inversa
—u	tira uma cópia das linhas iguais
+ pos1[—pos2]	a chave de classificação começa na posição 1 e finaliza na 2
—o nome	usa 'nome' como arquivo de saída
arquivos...	um ou vários arquivos

SU — **entrada em super-usuário**
su [—] [nome] [argumento]

—	muda os parâmetros do usuário como se estivéssemos entrando em um novo
nome	nome do usuário
argumentos	argumentos para a shell

SYNC — **atualiza periodicamente o super-bloco**
sync

TAIL — **lista a última parte de um arquivo**
tail [+—[contador] [lbc]] [arquivo]

+contador	número de linhas desde o começo do arquivo
—contador	número de linhas desde o fim do arquivo
b	contador de blocos
c	contador de caracteres
l	contador de linhas
arquivo	nome do arquivo

TAR — **efetua cópias em formato de arquivo (archive)**
tar [chave] [nomes de arquivo]

chave	controla a ação do comando; as chaves podem ser:
0... 7	seleciona o drive
c	cria uma nova cópia
l	visualiza mensagens de erro se existir algum link incorreto
m	não recupera arquivos que tenham sido modificado posteriormente
t	lista os nomes dos arquivos
v	lista nomes de arquivo seguidos por mais informação
w	espera confirmação de cada arquivo a copiar ou recuperar

TEST — **avalia a condição de um comando**
test [—b arquivo] [—c arquivo] [—d arquivo] [—f arquivo] [—g arquivo] [—k arquivo] [—ns1] [n1 — eqn2] [—p arquivo] [—r arquivo] [—s arquivo] [s1] [s1 = s2] [s1

!= s2] [—t fildes] [—u arquivo] [—w arquivo] [—x arquivo] [—z s1]

—b arquivo	verdadeiro se o arquivo existe e é um bloco especial do arquivo
—c arquivo	verdadeiro se o arquivo existe e é um caracter especial do arquivo
—d arquivo	verdadeiro se o arquivo existe e é um diretório
—f arquivo	verdadeiro se o arquivo existe e é regular
—g arquivo	verdadeiro se o arquivo existe e seu bit do set-grupo IB é trocado
—k arquivo	verdadeiro se o arquivo existe e seu bit de sticky é trocado
—ns1	verdadeiro se o comprimento do strint (s1) não é 0
n1 —eqn2	verdadeiro se os inteiros n1 e n2 são iguais
—p arquivo	verdadeiro se o arquivo existe e é chamado pipe
—r arquivo	verdadeiro se o arquivo existe e é legível
—s arquivo	verdadeiro se o arquivo existe e seu tamanho é maior do que 0
s1	verdadeiro se a cadeia s1 não é nula
s1 = s2	verdadeiro se as cadeias s1 e s2 são iguais
s1 != s2	verdadeiro se as cadeias s1 e s2 não são iguais
—t fildes	verdadeiro se o número descritor do arquivo estiver associado ao terminal, por default é 1
—u arquivo	verdadeiro se o arquivo é 1 e seu bit do set-usuário ID está trocado
—w arquivo	verdadeiro se o arquivo existe e pode ser escrito
—x arquivo	verdadeiro se o arquivo existe e é executável
—z s1	verdadeiro se o comprimento da cadeia é 0

TIME — tempo de um comando

time comando

comando	comando a ser cronometrado

TRUE — provê os valores verdade

true
false

TTY — lista o nome completo de um terminal

tty [—l] [—s]

—s	suprime a impressão do nome do terminal; permite ver o código de saída

UMASK — troca a máscara da forma de criação de um arquivo

umask [ooo]

ooo	troca a máscara da forma de criação de um arquivo à ooo.

VI — editor orientado na tela

vi [—t tag] [—r] [+comando] [—l] [—wn] nome...

—t tag	equivalente a um comando inicial tag
—r	recupera a última versão salva do nome do arquivo especificado
+comando	começa a execução do comando especificado
—l	troca as opções showmatch e lisp do LISP
—wn	troca o tamanho por default da janela a "n"
nome...	um ou mais nomes de arquivos

O seguinte é uma lista dos comandos do editor VI. Cada comando da lista está agrupado de acordo com sua função de edição. Os comandos Ex podem ser executados no editor VI precedendo o comando com a:. Todos os comandos com a palavra CONTROL são mantidos pela chave CONTROL e precedem à chave indicada.

Comandos de cancelamento do VI:

Q	Termina o modo do editor VI
—ZZ	Escreve o buffer no arquivo atual e sai do editor

Comandos de movimento do cursor:

CONTROL-B ou b	move o cursor para trás ao princípio de uma palavra

CONTROL-C	volta à página anterior
CONTROL-D	baixa a tela a meia janela
CONTROL-E	escreve uma linha adicional ao final da tela em curso
CONTROL-F	tela seguinte
CONTROL-G	visualiza, se o arquivo foi modificado, o número da linha em curso, número de linhas no arquivo e porcentagem de localização
CONTROL-H ou h	move o cursor uma posição atrás
CONTROL-J	avança à linha seguinte, na mesma coluna
CONTROL-M	avança à linha seguinte
CONTROL-N ou j	à linha seguinte na mesma coluna
CONTROL-P ou k	à linha anterior na mesma coluna
CONTROL-U	move a tela para cima, meia janela
CONTROL-Y	visualiza uma linha adicional ao princípio da tela em curso
—B	move o cursor até o princípio de uma palavra
E ou e	move o cursor até o final da palavra em curso
Fx	localiza ou encontra o primeiro caracter "x"; não modifica a linha em curso
fx	localiza ou encontra o primeiro caracter "x" desde a linha em curso
G	vai ao número de linha especificado
H	move o cursor à primeira linha
L	move o cursor ao primeiro caracter da primeira linha
M	move o cursor à linha central da tela
N	inverte a ordem de busca
n	repete a última busca
Tx	localiza o caracter "x" depois do cursor e o coloca depois do caracter
tx	avança o cursor para cima, até o caracter posterior a "x"
W	move o cursor até o princípio da palavra seguinte na linha
w	move o cursor ao princípio da palavra seguinte
SPACE ou l	avança o cursor até o caracter seguinte na linha

+	avança o cursor ao primeiro caracter que não seja branco da linha seguinte
—	posiciona o cursor no primeiro caracter que não seja branco na linha anterior
/string	examina desde a ocorrência seguinte na cadeia
?string	examina até a ocorrência seguinte na cadeia
!	move o cursor até a posição do primeiro caracter não branco
;	repete o último caracter encontrado de f, F, t e T
S	move o cursor até o final da linha em curso
%	move a posição do cursor aos parênteses ou chaves cuja máscara está em curso
)	linha seguinte
(	linha anterior
}	parágrafo seguinte
{	parágrafo anterior
]]	seção seguinte
[[	seção anterior

Comandos de manipulação de texto do vi:

A	acrescenta ao final da linha
a	acrescenta depois da posição do cursor
C	troca o texto da linha em curso por outro
CONTROL-?	interrompe o editor para voltar ao estado de entrada do comando
D	elimina o texto desde a posição do cursor até o final da linha
d obj	elimina o objeto especificado
ESC	chave de escape para terminar uma entrada de A, a, C, c, S e s
I	insere ao princípio da linha
i	insere depois da posição do cursor
J	junta linhas
O	insere uma linha sobre a linha em curso

o	insere uma linha sob a linha em curso
P ou p	recupera o último texto eliminado depois/antes da posição do cursor
R	substitui caracteres
rc	substitui o caracter no qual está posicionado o cursor por c
S	substitui a linha pelas linhas introduzidas
s	substitui o caracter no qual está o cursor por uma cadeia de caracteres
U	deixa a linha em curso como estava antes de fazer mudanças
u	recupera a última mudança feita na linha
X	elimina o caracter anterior ao cursor
x	elimina o caracter no qual está o cursor
ny	copia "n" linhas num buffer, o recupera com P/p
	repete a última mudança

Comandos de ajuste da tela

CONTROL-L	limpa e volta a desenhar a tela
CONTROL-R	volta a desenhar a tela em curso, eliminando as linhas

Comandos diversos do vi:

CONTROL-I	quando inserido, imprime um número ou espaço como feito pela opção tabstop
CONTROL-Q ou V	utiliza inserções de não-impressão e caracteres especiais
CONTROL-S	suspende a saída até que seja introduzido outro CONTROL-S
CONTROL-W	move o cursor até o princípio da palavra anterior
CONTROL-[	cancela um comando com forma parcial
CONTROL-]	busca por um tag à partir da posição do cursor
CONTROL-|	volta à posição prévia no último arquivo editado
CONTROL-@	substitui o último texto inserido

m letra	marca a posição em curso com letra
0	move-se até o primeiro caracter da linha em curso
1-9	forma argumentos numéricos
:	pré-fixa o comando ex e manipula a opção
<	mudanças à esquerda
>	mudanças à direita
=	muda o escalonamento da linha do LISP
''	volta ao contexto anterior
!obj cmd	processa o objeto especificado no buffer
''nome	precedido de um nome de buffer; o número/nome pode ser desde 1 até 9 para salvar o texto eliminado e desde a até z para textos armazenados
# número	substitui o número por uma função no terminal
@	repete a substituição anterior
vi	opções de manipulação
autoindent	fornece escalonamento automaticamente, por default é noai
ignorecase	ignorá caso de busca; por default é noic
lisp	comandos ({) }i; por default é nolisp
list	o tabulador imprime como I, fim de linha marcado como $; por default é nolist
magic	os caracteres ".", "[" e "*" são especiais nas explorações; por default é nomagic
number	as linhas são impressas com um número de linhas pré-fixadas; por default é nonu
paragraphs	com os macros nomeados começa a dividir os parágrafos; por default é para = IPLPPPQPbpbPLI
redraw	simula um terminal inteligente, por default é nore
sections	para começar novas seções; por default é =NHSHH HU
shiftwidth	muda distância para < and >, por default é SW = 8

showmatch	mostra acompanhado de (or {as)or}; por default é nosm
slowopen	pospõe, mostrar fora de data enquanto que insere; por default é slow
term	especifica o tipo de terminal que está sendo utilizado
wrapscan	busca o fim do buffer se não for encontrado nenhum string; por default é ws

WC — conta linhas, palavras e caracteres em um arquivo

wc [—lwc] [arquivo...]

—c	conta somente caracteres
—l	conta somente linhas
—w	conta somente palavras
arquivo...	um ou mais nomes de arquivos

WHO — lista todos os usuários no sistema

who [—uTlpdbrtas] [arquivo] [am I]

—u	lista informação acerca dos usuários que estão no sistema
—T	se alguém pode ou não escrever nesse terminal; + se puder, — em caso contrário
—l	lista a linha na qual o sistema está esperando para que alguém entre com login
—p	lista os processamentos ativos atualmente, previamente produzidos pelo init
—d	imprime os processamentos terminados não produzidos pelo init
—b	mostra data e hora da última inicialização
—r	indica o nível de execução atual do processamento init
—t	indica a última mudança do root ao relógio do sistema
—s	lista somente nome, linha, hora
arquivo	examina o arquivo especificando em lugar do letc/utmp
am i	lista o nome do usuário

WHOAMI — lista o nome do usuário

whoami

WRITE — envia mensagens à outro usuário

write usuário [ttyname]

user	nome do usuário
ttyname	nome do terminal (se o nome do usuário estiver em mais de um terminal).

NOTAS

ÍNDICE GERAL

INFORMÁTICA

Desde sua aparição comercial no final dos anos 70, o sistema operacional UNIX foi convertido em um padrão de mercado para os modernos microcomputadores multi-usuário e multitarefa de 16 e 32 bits. O número de fabricantes e usuários que se servem dele nos mais diversos campos de aplicação está constantemente aumentando. Este volume da Biblioteca Básica Informática busca ser uma primeira introdução ao UNIX, descrevendo suas origens, estrutura, elementos e possibilidades de uma maneira simples, explicando claramente os conceitos básicos e facilitando ao leitor seu primeiro contato com o UNIX e C, de forma que se encontre em condições, se o deseja, de aceder a outras obras mais especializadas.

Cz$ 65,00

B.B.L.

UNIX – O FUTURO DOS SISTEMAS OPERACIONAIS

34