Módulo 8

Sistema Operacional

Lição 3

Comandos Básicos e Scripting

Versão 1.0 - Mar/2008

Autor
-

Equipe
Rommel Feria
John Paul Petines

Necessidades para os Exercícios

Sistemas Operacionais Suportados

NetBeans IDE 5.5 para os seguintes sistemas operacionais:

- Microsoft Windows XP Profissional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

NetBeans Enterprise Pack, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Profissional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

Configuração Mínima de Hardware

Nota: IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

Configuração Recomendada de Hardware

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

Requerimentos de Software

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris**, **Windows**, e **Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em http://java.sun.com/j2se/1.5.0/download.html
- Para **Mac OS X**, Java 2 Plataform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: http://developer.apple.com/java (é necessário registrar o download da JDK).

Para mais informações: http://www.netbeans.org/community/releases/55/relnotes.html

Colaboradores que auxiliaram no processo de tradução e revisão

Aécio Júnior
Alberto Ivo da Costa Vieira
Alexandre Mori
Alexis da Rocha Silva
Allan Wojcik da Silva
Antonio José Rodrigues Alves Ramos
Angelo de Oliveira
Bruno da Silva Bonfim
Carlos Fernandes Gonçalves
Denis Mitsuo Nakasaki
Felipe Gaúcho
Jacqueline Susann Barbosa
João Vianney Barrozo Costa
Luiz Fernandes de Oliveira Junior
Marco Aurélio Martins Bessa
Maria Carolina Ferreira da Silva
Massimiliano Giroldi
Paulo Oliveira Sampaio Reis
Ronie Dotzlaw
Seire Pareja
Thiago Magela Rodrigues Dias
Vinícius Gadis Ribeiro

Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos -** Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo -** Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Rodrigo Nunes -** Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso -** Coordenador responsável pelo ambiente JEDI™ (Moodle)

Agradecimento Especial

John Paul Petines – Criador da Iniciativa JEDI™
Rommel Feria – Criador da Iniciativa JEDI™

Original desta por – McDougall e Mauro – Solaris Internals. Sun Microsystems. 2007.

1. Objetivos

A aprendizagem de um sistema operacional vai além do conhecimento da interface gráfica. A interface gráfica é o bastante para usuários comuns, mas o terminal permite a execução de comandos mais poderosos. Nessa lição veremos alguns comandos básicos do terminal do Solaris e também uma introdução sobre *shell scripting*.

Ao final desta lição, o estudante será capaz de:

- Utilizar alguns comandos básicos em um Terminal
- Discutir a criação de *scripts* básicos e avançados
- Executar comandos básicos para administração do ambiente

2. Terminal

Podemos abrir um terminal pressionando o botão direito no *desktop* e selecionando a opção 'Open Terminal'. O *prompt* é onde os comandos serão executados. Nos exemplos mostrados, iremos representar o comando de terminal pelo símbolo $. O símbolo $ não deve ser digitado.

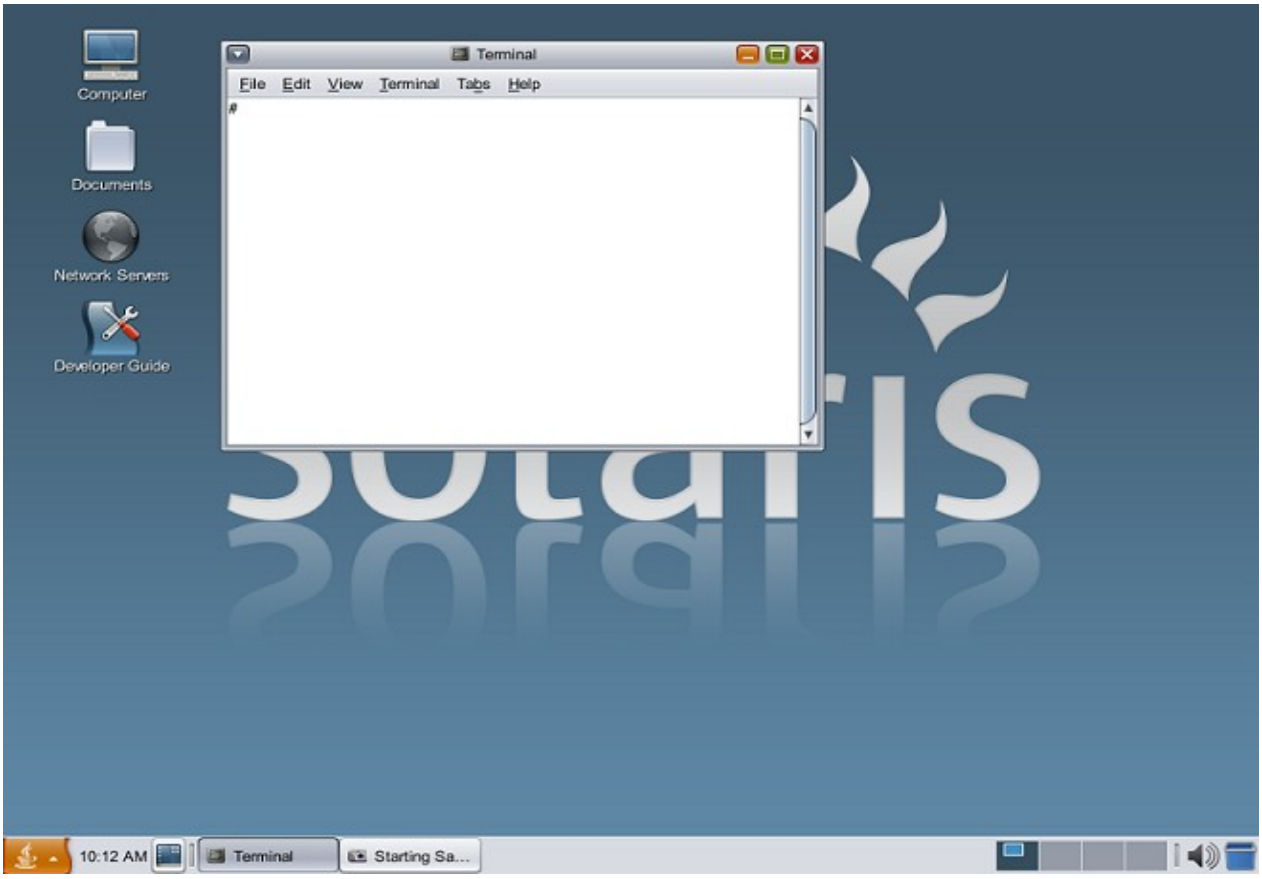

Figura 1: Janela de Terminal no Solaris

Mais à frente nessa lição, será solicitado a troca para o usuário definido por padrão como *root*. O usuário *root*, também conhecido como superusuário, é o administrador do sistema, com acesso a comandos que afetam a todo o sistema. O *prompt* do terminal para o superusuário muda de forma a refletir isso. Para representar o terminal do superusuário, usaremos o símbolo #. De modo semelhante, não devemos digitar o símbolo # inicial.

Comandos e nomes de arquivos um ambiente Unix, como o caso do Solaris e Linux, são sensíveis a letras maiúsculas e minúsculas. Ao executar, por exemplo, o comando para mostrar os arquivos de um diretório como **Ls**, ao invés de **ls**, será mostrado um erro informando que o comando não existe.

2.1. Comandos de ajuda

Cada comando no ambiente terminal possui uma documentação de ajuda. Podemos acessar essa ajuda através do comando **man** (***man****ual*) seguido pelo comando. Por exemplo, para solicitar a ajuda sobre o comando **ls**.

```
$ man ls
```

Podemos navegar pelo editor de ajuda utilizando as teclas para cima e para baixo. Para finalizar o editor de ajuda, pressionamos a tecla **q** (***q****uit*).

2.2. Navegar no sistema de arquivos

2.2.1. Visualizar os arquivos do diretório Atual

O comando **ls** (***l**ist **s**ources*) lista arquivos no diretório corrente. Podemos também mostrar os arquivos selecionados utilizando o comando **ls**. Da seguinte forma:

```
ls jedi.txt
```

Se nenhum arquivo no diretório corrente tiver o nome *jedi.txt*, então o comando não produz nenhuma saída.

O comando **ls** permite a utilização de curingas para visualizar arquivos que atendam a um certo critério. Apresentamos a seguir alguns curingas mais utilizados:

- ***** - representa nenhum ou vários caracteres. Por exemplo:
  - **ls a*** - mostrar os arquivos iniciados pela letra **a**
  - **ls *.java** - listar os arquivos que terminem com .java
  - **ls *** - listar os arquivos, neste caso o curinga é desnecessário pois este é o comportamento padrão deste comando
- **?** - representa um único caractere. Por exemplo:
  - **ls ?a?** - mostrar os arquivos que contém uma letra **a** no meio de um nome

Como a maioria dos comandos, o comando **ls** pode ter opções adicionais, que são iniciadas com o símbolo '-' (sinal de menos). Por exemplo, para visualizar arquivos no formato longo, que mostra informações adicionais do arquivo, devemos usar a opção **-l**. Para ver todos os arquivos (incluindo arquivos ocultos) podemos usar a opção **-a**. Para ambas, podemos digitar:

```
$ ls -a -l
```

Ou então:

```
$ ls -al
```

Executar o comando **ls** com a opção **-a** faz com que os arquivos ocultos sejam incluídos na listagem. Arquivo ocultos são arquivos cujo o nome é iniciado por um ponto. Por exemplo, o arquivo **.bashrc** é um arquivo oculto, um *script* que é executado sempre que uma janela de terminal é aberta. Podemos criar arquivos ocultos simplesmente utilizando um ponto no inicio do nome do arquivo.

Algumas opções consistem de palavras ao invés de letras simples. Algumas opções podem necessitar de parâmetros, os quais são especificados por um símbolo '=' (sinal de igual). O seguinte exemplo que mostra todos os arquivos, utiliza a opção **–sort**, com o parâmetro **size**, para ordenar pelo tamanho dos arquivos.

```
$ ls -al –sort=size
```

Existem muitas outras opções para o comando **ls** que podem ser verificadas utilizando o comando **man**.

2.2.2. Permissões de arquivos

Uma breve descrição da saída do comando **ls -l** é necessária. Em particular, iremos discutir permissões de arquivos.

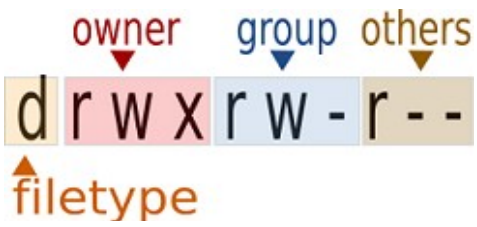

Figura 2: Sistema de Permissões

O primeiro espaço na lista de controle de acesso descreve o tipo do arquivo que está sendo listado:

- Um espaço em branco indica que é um arquivo comum
- Um **d** indica um diretório
- Um **I** ou **s** indica um caminho simbólico, do tipo *hardware* ou *software*, para outro arquivo, respectivamente
- Exclusivamente para dispositivos, um **b** indica que o arquivo representa um dispositivo de bloco enquanto que um **c** indica que o arquivo aponta para um dispositivo de caractere

Existem três permissões possíveis para as listas de controle:

- Para um arquivo ser lido, deve possuir a permissão **r**
- Para um arquivo ser alterado, deve possuir a permissão **w**
- Para um arquivo ser executado, deve possuir a permissão **x** (em ambiente tipo Unix não existe a diferença entre um arquivo executável é um arquivo texto, diferente de ambiente tipo DOS)

A lista de controle de acesso é dividida em três partes:

- Os primeiros três caracteres referem-se ao usuário que criou o arquivo
- Os próximos três caracteres contém a permissão do grupo de usuários que pertencem ao mesmo grupo do usuário que criou o arquivo
- As últimas três letras indicam as permissões para usuários que não estão no grupo do usuário que criou o arquivo

Considere as seguintes permissões de acesso para um arquivo qualquer: rwxrw-r--

As primeiras três letras determinam as permissões do usuário. O **r** nesse grupo de letras significa que o arquivo é legível pelo criador do arquivo. O **w** que o arquivo pode ter seu conteúdo modificado (não sendo apenas de leitura). A permissão **x** significa que o arquivo pode ser executado. No exemplo, o usuário que criou o arquivo pode ler, escrever e executar este.

Usuários nos sistemas UNIX podem pertencer a grupos de usuários. Os próximos três caracteres determinam quais permissões outros usuários que pertencem ao mesmo grupo do criador do arquivo possuem. No exemplo, a permissão ausente, caractere **x**, significa que os outros usuários do grupo não podem executar o arquivo.

Para finalizar, os últimos três caracteres determinam as permissões para os usuários que não pertencem ao grupo do criador do arquivo. No exemplo, podem apenas ler o arquivo.

Permissões de arquivos podem ser modificadas com o comando **chmod** (***ch****ange* ***mod****e*). Aceita dois parâmetros, o nome do arquivo a ser modificado e a nova permissão utilizando três dígitos em formato **octal**.

Se um digito **octal** for convertido para **binário**, então o número **binário** consiste de três dígitos. O comando **chmod** necessita de três dígitos em formato octal:

- O primeiro para a permissão do usuário
- O segundo para permissões do grupo
- O terceiro para permissões de outros usuários.

Na montagem final da lista de controle de acesso convertendo o conjunto para binários, o binário **1** significa que a permissão é concedida e o binário **0** que não existe permissão para a posição.

Por exemplo, para um arquivo **hello.txt**, a permissão **rwxrw-r** pode ser configurada usando:

```
$ chmod 764 hello.txt
```

A seguir temos uma tabela para a conversão do tipos:

Octal	Binário	Permissões
0	000	---

1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Figura 3: Conversão Binário, Octal e Permissões

E a permissão **rw-r-xr--** pode ser configura usando:

```
$ chmod 654 hello.txt
```

2.2.3. Obter a localização atual e mudar de diretório

Ao entrar no terminal, por padrão, o diretório corrente é o *home*. Para descobrir qual é o diretório corrente, usamos o comando **pwd** (***p****resent* ***w****ork* ***d****irectory*):

```
$ pwd
/export/home/alice
```

O sistema de arquivo do Solaris começa no diretório **root**, ou diretório **/**. Para alcançar o diretório **home** do usuário logado a partir do diretório **/**, primeiramente devemos acessar o diretório **export**, e em seguida o diretório **home**, e nesse diretório acessar o diretório do usuário logado (que possui o mesmo nome deste).

Para mudar o diretório corrente, utilizamos o comando **cd** (***c****hange* ***d****iretory*) seguido do nome do diretório que deve ser acessado. Como não existe nenhum diretório dentro do diretório **home**, primeiramente devemos ir para o diretório **/**, conforme o comando a seguir:

```
$ cd /
```

Ao executar o comando **ls** nesse diretório, serão mostrado todos os diretórios e arquivos que estão no nível mais alto dos diretórios do sistema. A figura a seguir mostra um sistema de arquivo típico para sistemas baseados em UNIX, tal como o Solaris.

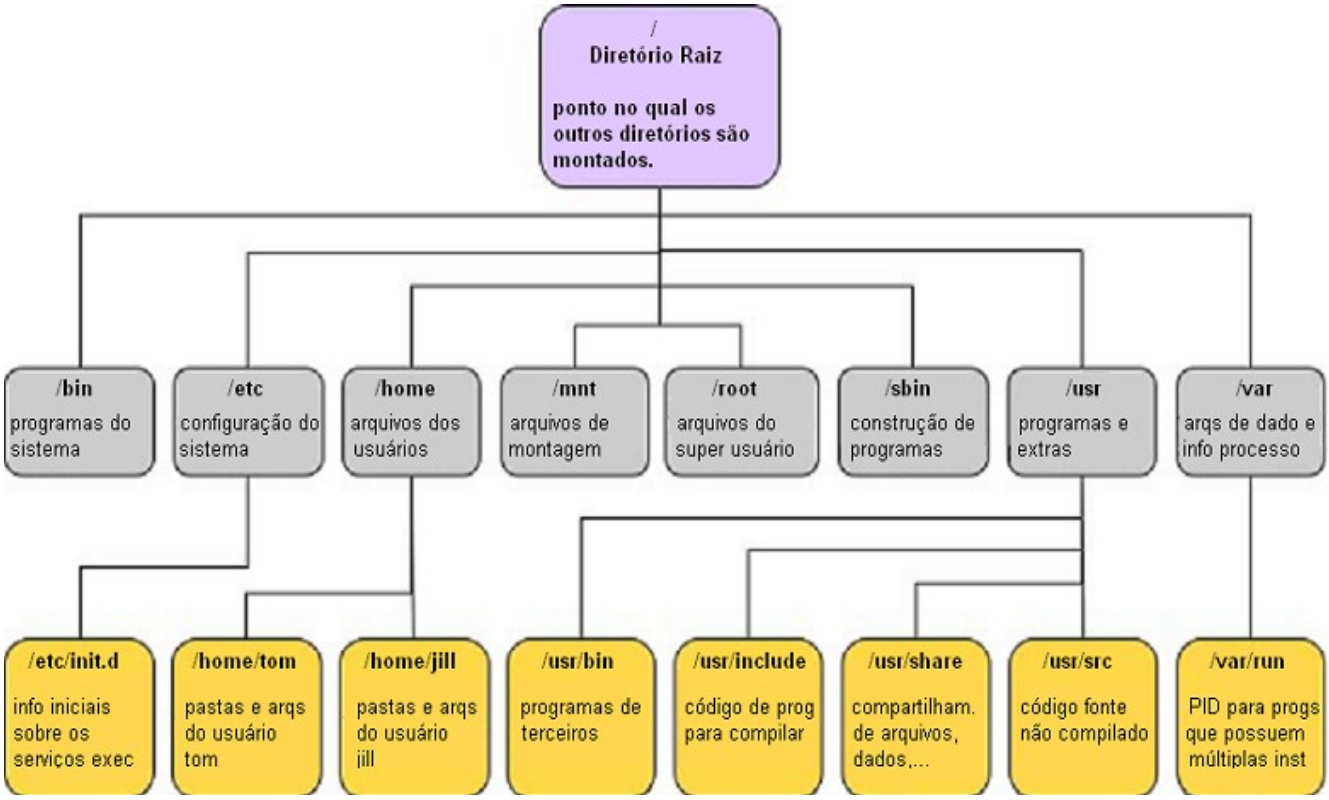

Figura 4: Sistema de arquivos padrão

Por exemplo, para acessar o diretório **etc**, a partir da raiz, podemos digitar:

```
$ cd etc
```

Ao executar o comando **pwd**, será mostrado o caminho do diretório **etc**. Como forma de exercitar estes comandos, entre nos subdiretórios de **etc** e verifique quais são seus arquivos.

Para subir um nível de diretório, podemos digitar o seguinte comando:

```
$ cd ..
```

Ponto seguido de ponto é uma referência para o diretório pai do diretório atual. Sendo assim, se o usuário estiver, por exemplo, em **etc**, poderá digitar esse comando para retornar ao diretório **/**.

Há duas maneiras para especificar um diretório. Podemos especificar seu nome relativo. Por exemplo, para navegar do diretório **/** ao diretório **defaults**, que está dentro do diretório **etc**, podemos digitar:

```
$ cd etc/defaults
```

Podemos também especificar o nome absoluto de um diretório, que é o caminho completo do diretório sempre a partir da raiz. Por exemplo, estando em qualquer diretório do sistema, o usuário pode retornar ao diretório **home** através do comando:

```
$ cd /export/home/<username>
```

Se o nome do usuário for "alice", então o comando seria da seguinte forma:

```
$ cd /export/home/alice
```

Observe que digitando apenas o comando **cd** fará com que o diretório **home** se torne o diretório corrente.

2.2.4. Sistema de arquivos Solaris

Voltemos à raiz do sistema Solaris. Embora existam muitos diretórios que podem mudar de acordo com a instalação, os seguintes diretórios são padrões:

- **/export/home** – O diretório **home** para todos os usuários. Pode também ser apenas **/home**
- **/usr** – Arquivos executáveis do sistema
- **/etc** – Arquivos de configuração
- **/var** – Diretório para arquivos temporários
- **/proc** – Diretório especial do sistema de arquivos para mostrar todas as aplicações que estão rodando no momento
- **/mnt** – Diretório onde as mídias removíveis são montadas, tais como, disquetes, CDs e pen drives
- **/dev** – Dispositivos são representados como arquivos no sistema Solaris e contidos neste diretório

2.2.5. Obter a utilização e o espaço livre no disco

O comando **du** (***d****isk* ***u****sage*) mostra quanto espaço está sendo gasto pelos arquivos num determinado diretório, incluindo sub-diretórios. Geralmente é usado com o atributo **-h**, para mostrar uma listagem mais completa.

```
$ du -h /etc
   5K   /etc/certs
   5K   /etc/cron.d
   5K   /etc/crypto/certs
   1K   /etc/crypto/crls
  11K   /etc/crypto
  63K   /etc/default
```

```
   3K   /etc/devices
...
56M /etc
```

O comando **df** (***d****isk* ***f****ree*) mostra, para cada disco, sua capacidade, quanto está em uso e quanto está livre. Assim como o comando **du**, o comando **df** também é, normalmente, utilizado com a opção **-h**.

```
$ df -h
Filesystem             size   used  avail capacity  Mounted on
/dev/dsk/c0t0d0s0       13G   3.8G   9.3G    30%    /
/devices                 0K     0K     0K     0%    /devices
ctfs                     0K     0K     0K     0%    /system/contract
proc                     0K     0K     0K     0%    /proc
mnttab                   0K     0K     0K     0%    /etc/mnttab
swap                   2.4G   1.1M   2.4G     1%    /etc/svc/volatile
objfs                    0K     0K     0K     0%    /system/object
fd                       0K     0K     0K     0%    /dev/fd
swap                   2.4G     0K   2.4G     0%    /tmp
swap                   2.4G    48K   2.4G     1%    /var/run
/dev/dsk/c0t0d0s7       19G   8.1G    11G    43%    /export/home
```

2.2.6. Localizar arquivos

Para procurar um arquivo, o usuário pode utilizar o comando **find** o qual procura pelo nome do arquivo (especificado com a opção **-name**) de um dado diretório, e, recursivamente, por todos os subdiretórios. É permitido o uso de curingas neste comando.

O exemplo abaixo procura todos os arquivos que começam pela palavra **profile** e estão dentro do diretório **/etc**.

```
$ find /etc -name profile*
```

Existem muitas opções para se usar juntamente com este comando. Para maiores informações, basta rodar o comando **man**.

2.3. Modificar o sistema de arquivos

Usuários podem fazer mudanças apenas em seus diretórios. Somente o usuário **root** pode fazer mudanças nos diretórios do sistema.

2.3.1. Copiar arquivos

O usuário pode copiar arquivos através do comando **cp** (***co****p****y***), o qual possui dois argumentos: o arquivo fonte e o de destino. Lembrando que pode conter diretórios e curingas nos nomes dos arquivos.

Por exemplo, para copiar o arquivo **passwd** do diretório **/etc** para o diretório atual, executamos:

```
$ cp /etc/passwd .
```

No comando o "." é uma referência ao diretório atual. E para copiar todos os arquivos do diretório **/etc** para o diretório /**home**:

```
$ cp /etc/* /export/home/<username>
```

Se o destino for um nome de arquivo e não um diretório, o comando **cp** copia o arquivo e o renomeia. Por exemplo, para um arquivo chamado **a.txt**, podemos realizar uma cópia para um arquivo chamado **b.txt**:

```
$ cp a.txt b.txt
```

Também podem ser feitas cópias de diretórios inteiros, usando a opção **-r** (***r****ecursive*). Cópias recursivas incluem subdiretórios do diretório especificado. Por exemplo, para copiar todo o conteúdo do **/etc** para o diretório **home** do usuário:

```
$ cp -r /etc /export/home/<username>
```

Uma vez copiado, aparecerá um novo diretório **/etc** no diretório **/home**.

2.3.2. Mover arquivos

Para mover arquivos, usamos o comando **mv** (***mo****v****e***) que contém opções e funcionalidades similares ao comando **cp**, só que ao término de uma cópia bem sucedida, o comando exclui os arquivos ou diretórios originais.

Por exemplo, para mover o arquivo **a.txt** para **c.txt**:

```
$ mv a.txt c.txt
```

2.3.3. Eliminar arquivos

Para eliminar arquivos, usamos o comando **rm** (***re****m****o****ve*) e o nome do arquivo a ser eliminado. Por exemplo, o seguinte comando elimina o arquivo **c.txt**:

```
$ rm c.txt
```

o comando **rm** pode ser usado com curingas. Por exemplo, para remover todos os arquivos que estão em um diretório tempo, dentro do diretório **home** do usuário:

```
$ rm /export/home/<username>/temp/*
```

Note que isto remove apenas arquivos. A remoção de diretórios será discutida a seguir.

2.3.4. Criar e eliminar diretórios

Para criar diretórios, usamos o comando **mkdir** (***ma****k****e*** ***dir****ectory*). Por exemplo, criar um diretório **lesson1**:

```
$ mkdir lesson1
```

Também é possível especificar o caminho completo, a partir da raiz. Por exemplo, criar um subdiretório **exercise1** dentro do diretório **lesson1**:

```
$ mkdir /export/home/<username>/lesson1/exercise1
```

Para eliminar diretórios, usamos o comando **rmdir** (***re****m****o****ve*** ***dir****ectory*). Este comando permite usar o caminho relativo ou absoluto. Por exemplo, remover o diretório **exercise1**:

```
$ rmdir /export/home/<username>/lesson1/exercise1
```

Note que o comando **rmdir** só pode ser utilizado se o diretório já estiver vazio, sem arquivos. Para remover um diretório, devemos primeiro utilizar o comando **rm** e em seguida o comando **rmdir** para remover o diretório.

No entanto, existe uma solução para este problema. Usar o comando **rm** com as opções **-rf**. A opção **-r** indica recursividade, isso significa que todos subdiretórios serão removidos, e a opção **-f** é utilizada para forçar essa remoção, sem perguntas de confirmação. O seguinte exemplo remove o diretório **lesson1** com todo seu conteúdo sem questionar a ação.

```
$ rm -rf /export/home/<username>/lesson1
```

Note que este é um comando bastante perigoso. Se o usuário estiver no diretório raiz e rodar o comando '**rm -rf .**', pode remover todo o sistema de arquivos se tiver permissão para tal. E não existe nenhuma forma simples para desfazer essa ação no Solaris. Portanto, o usuário deve ter EXTREMO cuidado ao utilizar este comando e ter total certeza do que está fazendo.

2.4. Direcionar a saída de um comando para um arquivo

2.4.1. Redirecionamentos

As saídas dos comandos podem ser redirecionadas para arquivos. Por exemplo, a listagem completa do conteúdo do diretório **/etc** pode exceder a capacidade da tela, então, o usuário pode

salvar esta saída em um arquivo texto para ser visualizado através de um editor de texto. Para isto, basta usar o comando junto ao operador "**>**" (sinal de maior que) seguido do nome do arquivo texto onde a saída será armazenada. Por exemplo, para listar todos os arquivos do diretório **/etc** e colocar seu conteúdo no arquivo **list.txt**:

```
$ ls -l /etc > list.txt
```

Depois de executado, o usuário pode visualizá-lo através de qualquer editor de texto e até guardá-lo para uma análise posterior. Este redirecionamento funciona com qualquer comando.

O operador "**>**" sobrescreve o conteúdo antigo do arquivo destino. Para concatenar a saída para um arquivo que já possui um conteúdo, utilizamos o operador "**>>**". Por exemplo, adicionar a saída de um novo comando **ls** ao arquivo **list.txt** (criado no exemplo anterior), sem eliminar o conteúdo anterior:

```
$ ls -l /usr >> list.txt
```

O comando para concatenar é geralmente usado durante um processo de *log*, na qual a saída dos programas, tais como, mensagens do sistema operacional, são armazenadas por um longo período de tempo para uma eventual auditoria.

O comando **echo**, mostra (na tela ou em arquivo) o parâmetro passado. Por exemplo:

```
$ echo 'Hello world!'
Hello world!

$ echo 'Listing the contents of /etc' > list.txt
$ ls -l /etc >> list.txt
$ echo 'Listing the contents of /usr' >> list.txt
$ ls -l /usr >> list.txt
```

Para ignorar totalmente a saída de um comando, o usuário pode redirecioná-la para um arquivo especial chamado **/dev/null**. Este é um arquivo especial que simplesmente descarta qualquer saída enviada para ele. Suprimir uma saída de comando pode mudar a forma como um programa se comporta. Por exemplo, o seguinte comando suprime totalmente a saída do comando **ls**.

```
$ ls /etc > /dev/null 2> /dev/null
```

2.4.2. Error streams

Output stream e *Error stream* agem de formas diferentes nos sistemas UNIX. *Output stream* é a saída esperada pelo usuário enquanto que *Error stream* é a saída quando ocorre um determinado problema.

Linguagens de programação freqüentemente oferecem comandos para enviar informação. Java possue dois objetos diferentes: **System.out** para uma saída comum e **System.err** para uma saída com erros.

É possível ver em ação o e*rror stream* quando executamos o seguinte comando:

```
$ ls nosuchfile.txt > output.txt
ls: nosuchfile.txt: No such file found
```

O erro do comando **ls** é enviado para a saída padrão, e não para o arquivo **output.txt**. O operando "**>**" redireciona somente o *output stream* do comando. Se também for necessário redirecionar o *error stream*, utilizamos o operando "**2>**":

```
$ ls nosuchfile.txt > output.txt 2> error.txt
```

2.4.3. Paginar a saída de conteúdo

O comando **more** pode ser utilizado para visualizar conteúdo na tela. Este comando paralisa a saída de forma que seja possível ver uma tela cheia por vez. É possível concatenar o comando **more** com o comando **ls** utilizando o comando "**|**" (sinal de pipe).

```
$ ls -l /etc | more
```

O comando **more**, no entanto, é limitado podendo ser visualizado somente no sentido para a frente. O comando **less** é similar ao comando **more**, entretanto permite a visualização da saída em ambos sentidos, para frente e para trás.

```
$ ls -l /etc | less
```

2.5. Variáveis de ambiente

Variáveis de ambiente são definidas para uso pelo sistema operacional. As variáveis são identificadas pelo símbolo **$** (sinal de cifrão) em seu início. Por exemplo, a variável $PATH lista os diretórios nos quais o terminal procura por arquivos executáveis quando o usuário executa algum comando. O seguinte exemplo verifica o valor da variável $PATH:

```
$ echo $PATH
```

Outros exemplos de variáveis de ambiente são $HOME, $USER e $PWD que mostram o diretório padrão, o usuário atual e o diretório atual, respectivamente. Para descobrir seus valores, execute o comando **echo** no terminal. A seguir vemos como mostrar esses valores, e também como mesclar mensagens com as variáveis em um único comando **echo**:

```
$ echo 'Olá! Sou ' $USER '. ' $HOME ' é meu diretório padrão.'
```

Resultando, por exemplo:

```
Olá! Sou alice. /export/home/alice é meu diretório padrão.
```

É possível utilizar aspas simples ou duplas no comando **echo**. A diferença é que ao colocar uma variável dentro de aspas duplas, seu valor é retornado.

```
$ echo 'Olá! Meu nome é $USER'
Olá! Meu nome é $USER
$ echo "Olá! Meu nome é $USER"
Olá! Meu nome é alice.
```

Como pode ser visto, colocando o símbolo **$** entre aspas duplas no comando **echo** significa que as palavras sejam consideradas como variáveis. Para o símbolo **$** ser impresso, basta colocar um símbolo **** (sinal de barra contrária) antes do símbolo **$**, isto é, **\$**.

As variáveis também podem ser definidas. Por exemplo, para uma variável de boas-vindas definida pelo próprio usuário, basta digitar:

```
$ greeting='Bem Vindo!'
```

Note que não é necessário colocar o símbolo **$** se estiver atribuindo um valor a uma variável (cuidado pois o símbolo **$** exibido acima corresponde ao *prompt*). Também não poderá haver espaços entre os dois lados do operador de atribuição. Os nomes das variáveis são sensíveis a maiúsculas e minúsculas.

É possível utilizar as variáveis criadas da mesma forma que as variáveis de ambiente.

```
$ echo $greeting ' Como está ' $USER '?'
Bem Vindo! Como está alice?
```

É possível também mudar o valor das variáveis existentes. Por exemplo, se for desejável adicionar o diretório **/home** à variável $PATH, basta digitar:

```
$ PATH=$PATH:/export/home/<username>
```

Deve ser observado que as variáveis definidas pelo usuário são acessíveis somente dentro do terminal onde foram declaradas. Se houver outras janelas de terminal abertas, não será possível acessar estas variáveis definidas. Além disso, assim que o terminal é fechado, as variáveis criadas desaparecerão e as variáveis ambiente retornarão aos seus valores originais.

3. Scripts

Um *script* é um arquivo que pode conter diversas instruções seqüenciais para execução no terminal (comparando com o ambiente DOS, é um arquivo **.bat**). Algumas tarefas dos sistemas envolvem um encadeamento sucessivo de comandos simples e, ao colocá-los em um único *script*, economiza-se tempo e esforço ao executar um único comando.

O *script* vai além de uma simples cadeia de comandos. A maioria das linguagens de *script* tem suas próprias construções de programação, tais como sentenças com comandos de decisão e repetição, e podem aceitar entrada de dados do usuário.

Na interface de console existe o que chamamos de interpretador de comandos, conhecido como *shell*. Existem vários tipos de *shells*, tais como, **ksh** ou **korn shell**, ou **csh** (*c shell*), cada um possui uma sintaxe distinta. Utilizaremos aqui a linguagem denominada **bash**. **Bash** é uma sigla para *Bourne-again shell*, que é uma revisão da linguagem *Bourne shell*. Entretanto, o conhecimento aprendido pode ser aplicável a outros *shells*.

3.1. Criando um Bash Script

Em exemplos anteriores listamos o conteúdo dos diretórios **/etc** e **/usr** e enviamos para um arquivo chamado **list.txt**. Abaixo vemos um *script* que executa as instruções de uma forma conjunta. Pode ser utilizado em qualquer editor de texto para escrever este arquivo que receberá o nome de **myscript**.

```
#!/bin/bash
# this is my first bash script.
echo 'Listing the contents of /etc' > list.txt
ls -l /etc >> list.txt
echo 'Listing the contents of /usr' >> list.txt
ls -l /usr >> list.txt
```

Para executar este arquivo, devemos primeiro modificar suas permissões. Para descobrir quais são as permissões do **script**, usa-se o comando **ls -l**.

```
$ ls -l myscript
-rw-r--r--    1   alice alice 6     Nov 12 8:40 myscript
```

As permissões para **myscript** são de leitura e escrita para o proprietário do arquivo (alice) e somente de leitura para outros usuários. Para tornar o **script** executável, adicionamos a permissão através do comando **chmod**, da seguinte maneira:

```
$ chmod 755 myscript
$ ls -l myscript
-rwxr-xr-x    1   alice alice 6     Nov 12 8:42 myscript
```

Agora todos usuários possuem a permissão para executar o arquivo **myscript**. Para executar o *script*, digitamos:

```
$ ./myscript
```

3.1.1. Comentários

No arquivo **myscript**, vemos linhas que são iniciadas pelo símbolo **#**. Estas linhas são comentários. No entanto, a primeira linha, iniciada pelo símbolo **#!** do *script* indica que este é um script ***bash***, cujo o executável está no diretório **/bin**. Deve haver pelo menos um espaço entre o símbolo **#** e a primeira letra do comentário.

A primeira linha do *script bash* não é exatamente um comentário mas um indicador da linguagem de script a ser utilizada:

- #!/bin/bash - informa ao SO para executar o script utilizando o bash
- #!/bin/ksh - informa ao SO para executar o script utilizando o korn shell (outro tipo de linguagem de script)

3.1.2. *Bash Scripts* embutidos

A maioria dos sistemas baseados em UNIX utilizam *scripts* extensivamente durante sua execução. Por exemplo, quando o **bash** é iniciado durante o boot do sistema, ele executa comandos do **/etc/profile**. A variável $PATH e outras são definidas neste momento.

Quando o usuário realiza a entrada no sistema, os arquivos ocultos **.bash_profile**, **.bash_login** e **.profile** são lidos e executados. Quando um terminal é iniciado, são executados os comandos do *script* **.bashrc** do usuário corrente. Quando uma sessão é finalizada, o **bash** executa os comandos do arquivo **.bash_logout**.

3.2. Scripting avançado

Iremos agora descobrir como as variáveis, sentenças de decisão e de repetição podem ser utilizadas. Também veremos como conseguir entrada de dados pelo executor do *script*.

3.2.1. Substituição de variáveis

Conforme discutido anteriormente, uma variável é indicada pelo símbolo **$**. O símbolo **$** é um comando que indica que o valor da variável deve ser substituído naquela posição quando o comando for executado.

Por exemplo, considere os seguintes comandos

```
$ x=42
$ echo $x
```

A variável **x** contém o valor **42**. O comando **echo $x** é substituído internamente por **echo 42**. Como na maioria das vezes as variáveis são utilizadas desta maneira, é necessário lembrar que o símbolo **$** não aparece durante a atribuição de uma variável ou quando uma variável for exportada.

Esta substituição pode ser vista com o caractere de nova linha. Como na maioria das linguagens de programação, uma nova linha pode ser impressa com uma barra contrária e o caractere **n** (isto é, **\n**). Entretanto, colocar **\n** em uma **string** no comando **echo** não produz uma nova linha. Para se fazer isto, deve-se utilizar **$'\n'**. Será inserida uma nova linha na posição onde **\n** aparecer.

```
$ echo 'hello \n world'
hello \n world
$ echo 'hello' $'\n' 'world'
Hello
world
```

É possível utilizar variáveis para armazenar a saída de alguns programas colocando o símbolo "**'**" (sinal de aspas simples) no comando a ser executado. Por exemplo, o *script* a seguir armazena a saída do comando **ls /etc** em uma variável e a exibe.

```
#!/bin/bash
x='ls /etc'
echo "Our variable contains the following files"
echo $x
```

3.2.2. Variáveis regulares

As variáveis em *scripts* são declaradas de forma usual. Contudo, não são visíveis fora do escopo do *script* a menos que sejam exportadas. Por exemplo, pode-se notar no *script* **/etc/profile** que ele exporta a variável $PATH.

3.2.3. Variáveis posicionais da entrada de dados pelo usuário

As variáveis especiais **$1** a **$9** substituem argumentos no arquivo de *script*. Os argumentos são as palavras informadas e separadas por espaços após o nome do *script* (assim como na linguagem Java). Esta é uma das formas pelas quais o usuário pode entrar com dados. Vejamos o *script* abaixo, salvo em um arquivo chamado **argtest**. A quantidade de argumentos passados

para o *script* é representado pela variável **$#**.

```
#!/bin/bash
echo 'My first argument' $1
echo 'My second argument' $2
echo 'Number of arguments passed' $#
```

Abaixo estão alguns exemplos para o **argtest**.

```
$ ./argtest
My first argument
My second argument
Number of arguments passed 0

$ ./argtest hello
My first argument hello
My second argument
Number of arguments passed 1

$ ./argtest hello world star
My first argument hello
My second argument world
Number of arguments passed 3
```

Para variáveis posicionais acima de **$9**, o valor deverá ser colocado dentro de colchetes, por exemplo **${10}**, **${11}** e assim por diante.

3.2.4. Comando read

Podemos obter entradas para o *script* através do comando **read**. Por exemplo, o *script* abaixo solicita um nome:

```
#!/bin/bash
echo "Enter name"
read n
echo "Hello," $n "!"
```

3.2.5. Código de erro

Todos os comandos na maioria de sistemas UNIX possuem código de erro. O valor do código de erro varia entre 0 e 255. Por convenção, um programa retorna 0 após uma execução com sucesso. Qualquer outro valor informa que ocorreram problemas.

Em **bash**, para se obter o código de erro da última execução, utilizamos a variável **$?**. Por exemplo, o *script* a seguir mostra o código de erro do comando **ls** após a procura por um arquivo específico.

```
#!/bin/bash
ls $1
echo 'The errorcode of ls command is: ' $?
```

Abaixo temos a saída da execução (o arquivo foi salvo como **lstest**)

```
$ ./lstest
<list of files>
The errorcode of ls command is: 0
$ ./lstest nosuchfile.txt
No such file or directory
The errorcode of ls command is: 1
```

Especificamos o código de erro do *script* pelo comando **exit**. Por exemplo, no final do **lstest**, retornamos o código de erro do *script* como sendo o código de erro do comando **ls**. Observe que salvamos o valor da variável **$?** em outra variável, pois **$?** retorna o código de erro do último comando executado. Se não fosse salva, seria retornado o código de erro do comando **echo**.

```
#!/bin/bash
ls $1
```

```
output=$?
echo 'The errorcode of ls command is: ' $output
exit $output
```

3.2.6. Operadores

Os operadores aritméticos são usualmente **+**, **-**, ***** ou **/**. O operador **%** retorna o resto de uma divisão. O resultado de uma expressão aritmética pode ser atribuída a uma variável usando o comando **let**. Por exemplo:

```
$ x=5
$ let "x = $x + 1"
$ echo $x
6
```

Podemos utilizar também o operador **((**<expressão>**))**, que avalia a expressão dentro dos parênteses duplos. Note que o **$** encontra-se antes do abrir parênteses para ocorrer a substituição do valor.

```
$ x=$((5 + 5))
$ echo $x
10
```

A linguagem **bash** não utiliza números decimais e os transforma para valores do tipo *Strings*.

3.2.7. Estrutura condicional

Em **bash**, **0** é um valor verdadeiro e **1** representa falso. Isto reflete o valor de retorno de todos os comandos no Solaris. Por convenção, um programa retorna **0** após uma execução com sucesso e **1** caso contrário.

Podemos testar uma determinada condição utilizando a sentença **if**. Diferente de outras linguagens de programação, a condição lógica é envolvida por um conjunto de colchetes e o comando é encerrado com **fi**. De modo similar a outras linguagens de programação, podemos ter a declaração **if**, **if-else** e **if-elseif-else**. A seguir a sintaxe da sentença **if-elseif-else**:

```
if [condition] then
  <statements>
elseif [condition] then
  <statements>
else
  <statements>
fi
```

A seguir mostramos algumas condições que podem fazer parte da estrutura condicional **if**. Observe que "$a" e "$b" podem ser variáveis ou números.

Operador	Definição
`if ["$a" -eq "$b"]`	Igualdade entre números (igual a)
`if ["$a" = "$b"]`	Igualdade entre Strings (igual a)
`if ["$a -ne "$b"]`	Diferença entre números (não igual a)
`if ["$a" != "$b"]`	Diferença entre Strings (não igual a)
`-gt, -ge, -lt, -le`	Maior que, maior ou igual a, menor que, menor ou igual a
`-n, -z`	Comparação de não nulo ou nulo. Por exemplo, if [-n "$1"] verifica se o primeiro argumento não é nulo.
`if [<cond1>] && [<cond2>]`	Operador E
`if [<cond1>] || [<cond2>]`	Operador OU
`if [!<cond>]`	Operador de negação. Por exemplo, if [! "$a" -gt "$b"] significa que estamos perguntando se a negação do valor de a é maior que o valor de b
`-f`	Verifica se um nome especifico de arquivo existe. Por exemplo if [-f "hello.txt"] verifica se o arquivo hello.txt existe.
`-r, -w, -x`	verifica se o arquivo possui privilégios de leitura, escrita e execução.

Figura 5: Condições para o comando if

Por exemplo, retornando ao *script* **myscript**, podemos passar como argumento o nome do arquivo que desejamos salvar, o conteúdo de **/etc** e **/usr**. Se nenhum argumento for passado, isto é, **$1** será nulo, então devemos retornar um erro.

```
#!/bin/bash
# this is my first bash script.
if [-n $1] then
  echo 'Listing the contents of /etc' > $1
  ls -l /etc >> $1
  echo 'Listing the contents of /usr' >> $1
  ls -l /usr >> $1
else
  echo 'You should specify a parameter'
fi
```

3.2.8. Estruturas de repetição

Em linguagem **bash**, temos duas estruturas de repetição: **for** e **while**.

Estrutura de repetição for

A sintaxe do comando **for** é:

```
for <var> in <list> do
  <statements>
done
```

O parâmetro **list** é uma lista de valores, no qual cada valor desta lista será repassado para a variável **var**. Na primeira interação, **var** assume o valor do primeiro elemento da lista, na segunda interação **var** assume o segundo elemento da lista e deste modo sucessivamente.

A seguir um exemplo do comando **for** que percorre os dias da semana.

```
for days in "Seg" "Ter" "Qua" "Qui" "Sex" "Sab" "Dom" do
  echo $days
done
```

Estrutura de repetição while

Realiza a interação enquanto uma condição for verdadeira. A sintaxe para o comando **while** é:

```
while [condition] do
  <statements>
done
```

Por exemplo, o código a seguir mostra o texto "hello world" um determinado número de vezes, baseado no argumento passado.

```
#!/bin/bash
y=0
while [$y -lt $1] do
  echo 'hello world'
  let "y = y + 1"
done
```

4. Comandos básicos de administração

4.1. Alterar para o super usuário: root

A administração do sistema só pode ser realizada pelo super usuário, isto é, o usuário **root**. Por exemplo, a edição dos arquivos *scripts* de profile de configuração no diretório **/etc** só pode ser feito pelo usuário **root**. Para acessar a conta **root**, devemos iniciar o sistema com este usuário ou utilizar o comando **su** (***s**ubstitute **u**ser* ou ***s**witch **u**ser*) para modificar o usuário.

```
$ su
Enter password: *******
#
```

Note que o sinal do *prompt* foi modificado para refletir que agora o usuário possui o *status* do super usuário. O comando **su** também pode ser utilizado para alterar para qualquer usuário.

```
$ su bob
Enter password: *******
$ (<-- bob é o usuário corrente)
```

4.2. Administração de usuários

A seguir veremos alguns comandos que o usuário *root* pode utilizar para administrar os usuários do sistema.

Adicionar novos usuários

Para adicionar novos usuários, utilizamos o comando **useradd** (***user add***). Por exemplo, a instrução a seguir adiciona o usuário "alice"

```
# useradd -d /export/home/alice -m -s /bin/bash alice
```

As opções adicionais são:

- **-d** especifica um diretório home para o usuário. Deve ser configurado em **/export/home**
- **-m** o diretório será criado manualmente
- **-s** especifica o tipo *shell* que será utilizado pelo usuário **alice**, nesse caso o **bash**

Eliminar usuários

Para eliminar usuários, utilizamos o comando **userdel** (***user del**ete*). Por exemplo, para remover o usuário "alice".

```
# userdel -r alice
```

As opções adicionais são:

- **-r** se deve ocorrer a remoção do diretório do usuário

Trocar a senha dos usuários

Para trocar a senha do usuário utilizamos o comando **passwd** (***passw**or**d***). Por exemplo, para trocar a senha do usuário "alice" digite:

```
# passwd alice
```

Se nenhum parâmetro for passado para o comando, então é trocada a senha do usuário corrente. Esta é a forma utilizada pelos usuários que não possuem privilégio de administradores do sistema para trocar suas senhas.

Resumidamente, usamos:

- useradd <username> - para criar novo determinado usuário
- userdel <username> - para eliminar um determinado usuário
- passwd <username> - para modificar a senha de um determinado usuário.

Parceiros que tornaram JEDI™ possível

Instituto CTS
Patrocinador do DFJUG.

Sun Microsystems
Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

Java Research and Development Center da Universidade das Filipinas
Criador da Iniciativa JEDI™.

DFJUG
Detentor dos direitos do JEDI™ nos países de língua portuguesa.

Politec
Suporte e apoio financeiro e logístico a todo o processo.

Instituto Gaudium
Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.