Apêndice A

Shell básico

Se você nunca brincou de shell e quer saber do que se trata, então estude este capítulo com atenção, digite os exemplos e tente fazer todos os exercícios. Se você já conhece o shell e está precisando de um refresco para a memória antes de mergulhar nos estudos deste livro, passeie por este capítulo e teste seus conhecimentos. Se você já leu todo o livro e acompanhou o conteúdo sem dificuldade, pode saltar este capítulo sem dó.

Apresentação

O que é o shell

O shell é o *prompt* da linha de comando do Unix e do Linux, é o servo que recebe os comandos digitados pelo usuário e os executa.

O shell é aquele que aparece logo após digitar-se a senha do usuário e entrar na tela preta. Ou na interface gráfica, ao clicar no ícone do Xterm, rxvt, Terminal ou Console.

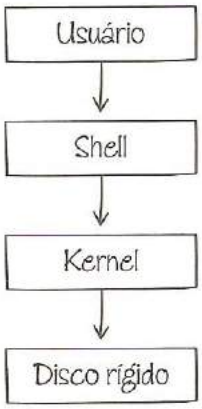
```
localhost login: root
Password:

Last login: Fri Apr 16 01:57:28 on tty5
[root@localhost root]# _
```

Ali está o shell, esperando ansiosamente por algum comando para ele poder executar. Essa é a sua função: esperar e executar. Cada comando digitado é lido, verificado, interpretado e enviado ao sistema operacional para ser de fato executado.

No Mac OS X, o shell está em Aplicativos > Utilitários > Terminal. No Windows é preciso instalá-lo com o Cygwin. Veja mais detalhes no tópico Shell no Linux, Mac e Windows, página 364.

Funcionando como uma ponte, o shell é a ligação entre o usuário e o kernel. O kernel é quem acessa os equipamentos (hardware) da máquina, como disco rígido, placa de vídeo e modem. Por exemplo, para o usuário ler um arquivo qualquer, toda esta hierarquia é seguida:



Para os usuários do Windows, é fácil pensar no shell como um MSDOS melhorado. Em vez do `C:\>` aparece um `[root@localhost root]#`, mas o funcionamento é similar:

- Basta digitar um comando, suas opções e apertar a ENTER que ele será executado.
- O comando deve estar no PATH.
- Mensagens de aviso são mandadas para a tela.
- Ctrl+C interrompe o funcionamento..

Isso tudo é igual em ambos. Mas o shell é muito mais poderoso que seu primo distante. Além dos comandos básicos para navegar entre diretórios e manipular arquivos, ele também possui todas as estruturas de uma linguagem de programação, como `IF`, `FOR`, `WHILE`, variáveis e funções. Com isso, também é possível usar o shell para fazer scripts e automatizar tarefas.

Este será o nosso foco: scripts em shell.

Shell script

Um script é um arquivo que guarda vários comandos e pode ser executado sempre que preciso. Os comandos de um script são exatamente os mesmos que se digita no prompt, é tudo shell.

Por exemplo, se de tempos em tempos você quer saber informações do sistema como horário, ocupação do disco e os usuários que estão logados, é preciso digitar três comandos:

```
[root@localhost root]# date
[root@localhost root]# df
[root@localhost root]# w
```

É melhor fazer um script chamado de `sistema` e colocar estes comandos nele. O conteúdo do arquivo sistema seria o seguinte:

```
#!/bin/bash
date
df
w
```

E para chamar este script, basta agora executar apenas um comando:

```
[root@localhost root]# sistema
```

Isso é um shell script. Um arquivo de texto que contém comandos do sistema e pode ser executado pelo usuário.

Antes de começar

Se você está acessando o sistema como usuário administrador (root), saia e entre como um usuário normal. É **muito perigoso** estudar shell usando o superusuário, você pode danificar o sistema com um comando errado.

Se você não tem certeza qual o seu usuário, use o comando `whoami` para saber.

Como o prompt de usuário normal é diferente para cada um, nos exemplos seguintes será usado `prompt$` para indicar o prompt da linha de comando.

O primeiro shell script

O primeiro shell script a fazer será o `sistema` do exemplo anterior, de simplesmente juntar três comandos em um mesmo script.

Passos para criar um shell script

1. Escolher um nome para o script

   Já temos um nome: sistema.

   Use apenas letras minúsculas e evite acentos, símbolos e espaço em branco.

2. Escolher o diretório onde colocar o script

   Para que o script possa ser executado de qualquer parte do sistema, mova-o para um diretório que esteja no seu PATH. Para ver quais são estes diretórios, use o comando:

   ```
   echo $PATH
   ```

   Se não tiver permissão de mover para um diretório do PATH, deixe-o dentro de seu diretório pessoal ($HOME).

3. Criar o arquivo e colocar nele os comandos

   Use o nano, VI ou outro editor de textos de sua preferência para pôr todos os comandos dentro do arquivo.

4. Colocar a chamada do shell na primeira linha

   A primeira linha do script deve ser:

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

Para que ao ser executado, o sistema saiba que é o shell quem irá interpretar estes comandos.

5. Tornar o script um arquivo executável

   Use o seguinte comando para que seu script seja reconhecido pelo sistema como um comando executável:

```
chmod +x sistema
```

Problemas na execução do script

Comando não encontrado

O shell não encontrou o seu script.

Verifique se o comando que você está chamando tem exatamente o mesmo nome do seu script. Lembre-se de que no Unix/Linux as letras maiúsculas e minúsculas são diferentes, então o comando `SISTEMA` é diferente do comando `sistema`.

Caso o nome esteja correto, verifique se ele está no `PATH` do sistema. O comando `echo $PATH` mostra quais são os diretórios conhecidos, mova seu script para dentro de um deles, ou chame-o passando o caminho completo.

Se o script estiver no diretório corrente, chame-o com um `./` na frente, assim:

```
prompt$ ./sistema
```

Caso contrário, especifique o caminho completo desde o diretório raiz:

```
prompt$ /tmp/scripts/sistema
```

Permissão negada

O shell encontrou seu script, mas ele não é executável.

Use o comando `chmod +x seu-script` para torná-lo um arquivo executável.

Erro de sintaxe

O shell encontrou e executou seu script, porém ele tem erros.

Um script só é executado quando sua sintaxe está 100% correta. Verifique os seus comandos, geralmente o erro é algum `IF` ou aspas que foram abertos e não foram fechados. A própria mensagem informa o número da linha onde o erro foi encontrado.

O primeiro shell script (melhorado)

Nesse ponto, você já sabe o básico necessário para fazer um script em shell do zero e executá-lo. Mas apenas colocar os comandos em um arquivo não torna este script útil. Vamos fazer algumas melhorias nele para que fique mais compreensível.

Melhorar a saída na tela

Executar os três comandos seguidos resulta em um bolo de texto na tela, misturando as informações e dificultando o entendimento. É preciso trabalhar um pouco a saída do script, tornando-a mais legível.

O comando `echo` serve para mostrar mensagens na tela. Que tal anunciar cada comando antes de executá-lo?

```
#!/bin/bash
echo "Data e Horário:"
date
echo
echo "Uso do disco:"
df
echo
echo "Usuários conectados:"
w
```

Para usar o `echo`, basta colocar o texto entre "aspas". Se nenhum texto for colocado, uma linha em branco é mostrada.

Interagir com o usuário

Para o script ficar mais completo, vamos colocar uma interação mínima com o usuário, pedindo uma confirmação antes de executar os comandos.

```
#!/bin/bash
echo "Vou buscar os dados do sistema. Posso continuar? [sn] "
read RESPOSTA
test "$RESPOSTA" = "n" && exit

echo "Data e Horário:"
date
echo
echo "Uso do disco:"
df
echo
echo "Usuários conectados:"
w
```

O comando read leu o que o usuário digitou e guardou na variável RESPOSTA. Logo em seguida, o comando test verificou se o conteúdo dessa variável era "n". Se afirmativo, o comando exit foi chamado e o script foi finalizado. Nessa linha há vários detalhes importantes:

- O conteúdo da variável é acessado colocando-se um cifrão "$" na frente.
- O comando test é muito útil para fazer vários tipos de verificações em textos e arquivos.
- O operador lógico && só executa o segundo comando caso o primeiro tenha sido OK. O operador inverso é o ||.

Melhorar o código do script

Com o tempo, o script vai crescer, mais comandos vão ser adicionados e quanto maior, mais difícil encontrar o ponto certo onde fazer a alteração ou corrigir algum erro.

Para poupar horas de estresse, e facilitar as manutenções futuras, é preciso deixar o código visualmente mais agradável e espaçado, e colocar comentários esclarecedores.

```
#!/bin/bash
# sistema - script que mostra informações sobre o sistema
# Autor: Fulano da Silva

# Pede uma confirmação do usuário antes de executar
echo "Vou buscar os dados do sistema. Posso continuar? [sn] "
read RESPOSTA

# Se ele digitou 'n', vamos interromper o script
test "$RESPOSTA" = "n" && exit

# O date mostra a data e a hora correntes
echo "Data e Horário:"
date
echo

# O df mostra as partições e quanto cada uma ocupa no disco
echo "Uso do disco:"
df
echo

# O w mostra os usuários que estão conectados nesta máquina
echo "Usuários conectados:"
w
```

Basta iniciar a linha com um # e escrever o texto do comentário em seguida. Estas linhas são ignoradas pelo shell durante a execução. O cabeçalho com informações sobre o script e seu autor também é importante para ter-se uma visão geral do que o script faz, sem precisar decifrar seu código.

Também é possível colocar comentários no meio da linha, `# como este`

Rebobinando a fita

Agora é hora de fixar alguns dos conceitos vistos no script anterior.

Variáveis

As variáveis são a base de qualquer script. É dentro delas que os dados obtidos durante a execução do script serão armazenados. Para definir uma variável, basta usar o sinal de igual "=" e para ver seu valor, usa-se o `echo`:

```
prompt$ VARIAVEL="um dois tres"
prompt$ echo $VARIAVEL
um dois tres
prompt$ echo $VARIAVEL $VARIAVEL
um dois tres um dois tres
prompt$
```

Não pode haver espaços ao redor do igual.

Ainda é possível armazenar a saída de um comando dentro de uma variável. Em vez de aspas, o comando deve ser colocado entre `$(...)`, veja:

```
prompt$ HOJE=$(date)
prompt$ echo "Hoje é: $HOJE"
Hoje é: Sáb Abr 24 18:40:00 BRT 2004
prompt$ unset HOJE
prompt$ echo $HOJE

prompt$
```

E, finalmente, o comando unset apaga uma variável.

Para ver quais as variáveis que o shell já define por padrão, use o comando env.

Detalhes sobre os comandos

Diferente de outras linguagens de programação, o shell não usa os parênteses para separar o comando de seus argumentos, mas, sim o espaço em branco. O formato de um comando é sempre:

```
COMANDO  OPÇÕES  PARÂMETROS
```

O comando `cat` mostra o conteúdo de um arquivo. O comando `cat -n sistema` mostra o nosso script, com as linhas numeradas. O `-n` é a opção para o comando, que o instrui a numerar linhas, e `sistema` é o último argumento, o nome do arquivo.

O `read` é um comando do próprio shell, já o `date` é um executável do sistema. Dentro de um script, não faz diferença usar um ou outro, pois o shell sabe como executar ambos. Assim, toda a gama de comandos disponíveis no Unix/Linux pode ser usada em scripts!

Há vários comandos que foram feitos para serem usados com o shell, são como ferramentas. Alguns deles:

Comando	Função	Opções úteis
`cat`	Mostra arquivo	`-n, -s`
`cut`	Extrai campo	`-d -f, -c`
`date`	Mostra data	`-d, +'...'`
`find`	Encontra arquivos	`-name, -iname, -type f, -exec`
`grep`	Encontra texto	`-i, -v, -r, -qs, -w -x`
`head`	Mostra início	`-n, -c`
`printf`	Mostra texto	nenhuma
`rev`	Inverte texto	nenhuma
`sed`	Edita texto	`-n, s/isso/aquilo/, d`
`seq`	Conta números	`-s, -f`
`sort`	Ordena texto	`-n, -f, -r, -k -t, -o`
`tail`	Mostra final	`-n, -c, -f`
`tr`	Transforma texto	`-d, -s, A-Z a-z`
`uniq`	Remove duplicatas	`-i, -d, -u`
`wc`	Conta letras	`-c, -w, -l, -L`

Use `man` *comando* ou *comando* `--help` para obter mais informações sobre cada um deles.

E o melhor, em shell é possível combinar comandos, aplicando-os em sequência para formar um comando completo. Usando o pipe | é possível canalizar a saída de um comando diretamente para a entrada de outro, fazendo uma cadeia de comandos. Exemplo:

```
prompt$ cat /etc/passwd | grep root | cut -c1-10
root:x:0:0
operator:x
prompt$
```

O cat mostra o arquivo todo, o grep pega essa saída e extrai apenas as linhas que contêm a palavra root e o cut, por sua vez, somente nessas linhas que o grep achou, extrai os 10 primeiros caracteres. Isso funciona como uma estação de tratamento de água, onde ela entra suja, vai passando por vários filtros que vão tirando as impurezas e sai limpa no final.

E, por fim, também é possível redirecionar a saída de um comando para um arquivo em vez da tela, usando o operador >. Para guardar a saída do comando anterior no arquivo saida, basta fazer:

```
prompt$ cat /etc/passwd | grep root | cut -c1-10 > saida
prompt$ cat saida
root:x:0:0
operator:x
prompt$
```

Cuidado, shell é tão legal que vicia!

O comando test

O canivete suíço dos comandos do shell é o test, que consegue fazer vários tipos de testes em números, textos e arquivos. Ele possui várias opções para indicar que tipo de teste será feito, algumas delas:

Testes em variáveis	
Opção	**Descrição**
-lt	Número é menor que (LessThan).
-gt	Número é maior que (GreaterThan).
-le	Número é menor igual (LessEqual).
-ge	Número é maior igual (GreaterEqual).
-eq	Número é igual (EQual).

-ne	Número é diferente (NotEqual).
=	String é igual.
!=	String é diferente.
-n	String é não nula.
-z	String é nula.
Testes em arquivos	
Opção	**Descrição**
-d	É um diretório.
-f	É um arquivo normal.
-r	O arquivo tem permissão de leitura.
-s	O tamanho do arquivo é maior que zero.
-w	O arquivo tem permissão de escrita.
-nt	O arquivo é mais recente (NewerThan).
-ot	O arquivo é mais antigo (OlderThan).
-ef	O arquivo é o mesmo (EqualFile).
-a	E lógico (AND).
-o	OU lógico (OR).

Tarefa: script que testa arquivos

Tente fazer o script `testa-arquivos`, que pede ao usuário para digitar um arquivo e testa se este arquivo existe. Se sim, diz se é um arquivo ou um diretório. Exemplo de uso:

```
prompt$ testa-arquivos
Digite o arquivo: /naoexiste
O arquivo '/naoexiste' não foi encontrado

prompt$ testa-arquivos
Digite o arquivo: /tmp
/tmp é um diretório

prompt$ testa-arquivos
Digite o arquivo: /etc/passwd
/etc/passwd é um arquivo

prompt$
```