# SHELL

## Descrição

É o interpretador de comandos do **Linux**.

## Informações complementares

- Na realidade, o *shell* é apenas um arquivo executável armazenado em */bin*. No modo gráfico, um *shell* é executado em cada terminal aberto.
- Os comandos digitados pelo usuário podem ser comandos internos (embutidos ou *builtin command*) do *shell*, mas na maioria das vezes eles são programas externos. A lista dos comandos embutidos pode ser obtida com o comando **help**.

```
help
```

  Os comandos não embutidos são programas que se iniciam invocando-se um **arquivo** executável em algum lugar no sistema de arquivos do Linux (o *shell* pesquisa em todos os diretórios listados na **variável de ambiente** PATH).

- O *shell* analisa sintaticamente a linha de comando depois que ela é lida. A **variável de ambiente** IFS (*Internal Field Separator*) determina como isso é feito. Normalmente, IFS é configurada de tal forma que espaços em branco separam os diferentes argumentos de linha de comando.
- Existem vários *shells* para o Linux, onde cada *shell* tem seus próprios recursos, capacidades e limitações.
- Para ver qual é o seu *shell* padrão, basta digitar o comando

```
printenv SHELL
```

  O comando acima exibe o conteúdo da **variável de ambiente** SHELL que contém o caminho completo do *shell.* Outra forma de saber qual é o *shell* padrão, é verificar o último parâmetro definido para o usuário no arquivo */etc/passwd.* Por exemplo,

```
aluno:x:501:501::/home/aluno:/bin/bash
```

  mostra que o usuário *aluno* usa o *shell bash*.

## Arquivos

Suponha que o *shell* padrão do sistema seja o *bash*. Então, quando o usuário acessa o sistema, o *bash* utiliza os arquivos abaixo do diretório *home* para montar o ambiente (estes arquivos são criados automaticamente pelo comando **adduser**):

1. **.bashrc** : contém funções, nomes alternativos (**alias**) e **variáveis de ambiente** do usuário.
2. **.bash_history** : contém a lista dos últimos (o padrão é 1000) comandos digitados pelo usuário.
3. **.bash_logout** : contém os comandos executados pelo sistema no fechamento da sessão pelo usuário.

Quando o usuário abre uma sessão, inicialmente são carregadas as definições gerais (armazenadas em */etc*) e, em seguida, as definições específicas do usuário (armazenadas em */home*).

Os arquivos do *shell bash* para os usuários estão em */etc/skel*. Estes arquivos são copiados para o diretório do usuário durante a criação da conta.

## Parâmetros

No *shell,* um parâmetro é uma entidade que pode armazenar um número, um nome ou um caractere especial. Quando o parâmetro é identificado por um nome, ele é uma variável.

Existem três tipos de parâmetros:

- parâmetros posicionais;
- parâmetro especiais;
- variáveis.

## Parâmetros posicionais

Um parâmetro posicional corresponde a um argumento fornecido na linha de comando. Ele é representado por $ seguido um ou mais dígitos. O menor valor para o dígito é 1, já que o parâmetro $0 corresponde ao nome do *script*. No caso da posição ser maior que 9, os dígitos deve ser colocados entre chaves.

Por exemplo, considere o *script shell* abaixo. Neste caso, o *script* informa quais são o primeiro, o segundo e o décimo parâmetros recebidos.

```
#!/bin/sh
echo “O nome do script = ” $0
echo “O primeiro argumento = ” $1
echo “O segundo argumento = ” $2
echo “O decimo argumento = ” ${10}
exit
```

Suponha que o nome do script seja *teste_param* e que doze parâmetros são fornecidos.

```
bash teste_param 1 oi a b c 7 8 9 teste xxx yyy zzz
```

A saída do programa é mostrada abaixo.

```
O nome do script = teste_param
O primeiro argumento = 1
O segundo argumento = oi
O decimo argumento = xxx
```

## Parâmetros especiais

Este tipo de parâmetro é tratado de forma especial pelo *shell.* O programador pode referenciar os parâmetros especiais, mas não pode modificá-los.

São exemplos:

- **$#** – contém o número de argumentos passado para o programa.
- **$*** – contém todos os argumentos passado para o programa.
- ***?** – contém informações sobre o último comando executado.
- **$$** – contém o número do processo em execução.
- **$!** – contém o número do processo do último comando executado em *background*.

Por exemplo, considere o *teste_param* com o seguinte código:

```
#!/bin/sh
echo “O número de argumentos = ” $#
echo “Argumentos passados = ” $*
exit
```

O *script* é então executado usando o comando abaixo.

```
bash teste_param 1 oi a b c 7 8 9 teste xxx yyy zzz
```

A saída do programa será:

```
O número de argumentos = 12
Argumentos passados = 1 oi a b c 7 8 9 teste xxx yyy zzz
```

# Caracteres especiais

O *shell* trata de forma especial as aspas, os apóstrofos e a barra invertida. Por exemplo, podemos digitar um dos quatro comandos abaixo

```
echo Guia Linux
echo “Guia Linux”
echo ‘Guia Linux’
echo Guia\ Linux
```

e o resultado será o mesmo

```
Guia Linux
```

Entretanto, quando um caractere especial é colocado entre as aspas, o *shell* interpreta este caractere. Assim, para saber o **PID** do *shell* sendo usado, podemos digitar

```
echo “pid = ” $$
```

ou

```
echo “pid = $$”
```

Teremos a mesma resposta do sistema. Mas se usarmos

```
echo ‘pid = $$’
```

a resposta será

```
pid = $$
```

pois, neste caso, o sistema não interpreta $$.

O *shell* não tenta interpretar o caractere que vem depois da barra invertida. Por exemplo,

```
echo \*
```

apenas mostra o caractere * na linha de comandos, enquanto

```
echo *
```

lista o conteúdo do diretório atual.

## Variáveis

Uma variável pode ser definida como:

```
nome=[valor]
```

Se um valor não é fornecido, a variável é associada a uma string nula (*null string*).

Um tipo especial de variável são as variáveis do shell que customizam o ambiente de trabalho do usuário. Por isso, elas são também conhecidas como **variáveis de ambiente**.

## Variáveis do shell

Os arquivos carregados pelo shell definem as variáveis de ambiente, que nada mais são que definições e valores que o shell e os outros programas do sistema reconhecem. Para ver quais as variáveis de ambiente no seu sistema você pode digitar **printenv** ou **env**. Por exemplo, são algumas das variáveis de ambiente do *bash*:

- **$** : número do processo do comando em execução.
- **CDPATH** : mostra uma lista separada por ":" que indica o caminho de busca para o comando **cd**. Por exemplo, "CDPATH=.:~:/etc" faz com que o sistema utilize a seguinte ordem de busca do diretório especificado: 1) a partir do diretório atual (representado por "."); 2) a partir do diretório raiz do usuário (representado por "~"); 3) a partir do diretório /etc.
- **FCEDIT** : define o editor padrão para editar comandos do "history". O padrão é o editor **vi**.
- **HISTFILE** : mostra o nome do arquivo que armazena as linhas de comando digitadas pelo usuário (no *shell bash* o arquivo padrão é o .bash_history).
- **HISTSIZE** : mostra o número de linhas de comando digitadas pelo usuário que são memorizadas pelo sistema.
- **HOME** : mostra o diretório *home* do usuário.
- **IFS** : separador de campos usado para definir como dividir as linhas em palavras para serem processadas separadamente. O valor padrão de IFS é *<space><tab><new-line>*.
- **LOGNAME** : mostra o nome de acesso do usuário.
- **MAIL** : mostra o diretório que contém as mensagens de correio eletrônicas recebidas pelo usuário.
- **OLDPWD** : mostra o diretório anterior de trabalho do usuário.
- **OSTYPE** : mostra o sistema operacional em uso.

- **PATH** : mostra caminho de busca dos comandos digitados pelo usuário.
- **PPID** : mostra o número de identificação do processo que inicializou o shell do usuário. Existe um diretório em */proc* com este número e que contém informações sobre o processo em questão.
- **PS1** : mostra a definição do *prompt* da linha de comando.
- **PS2** : mostra a definição do *prompt* secundário da linha de comando.
- **PWD** : mostra o diretório atual de trabalho do usuário.
- **SHELL** : mostra o nome do *shell* atualmente em uso.
- **SHLVL** : mostra o número de *shells* atualmente em execução na conta do usuário.
- **TERM** : mostra o tipo de terminal em uso.
- **TZ** : define o fuso horário a ser usado pelo sistema.
- **UID** : mostra o número de identificação do usuário.
- **USER** : mostra o nome do usuário atual.

É possível criar novas variáveis, excluir variáveis existentes ou apenas alterar o conteúdo de uma variável de ambiente.

Para criar uma nova variável de ambiente, basta definir o nome e o valor da nova variável de ambiente e usar o comando **export** para permitir que a variável seja visualizada pelos aplicativos (por exemplo, um novo *shell* ou um novo terminal) inicializados no mesmo terminal (neste caso a variável existirá enquanto a sessão estiver aberta). Por exemplo,

```
TESTE=10; export TESTE
```

ou

```
export TESTE=10
```

cria a variável de ambiente TESTE com valor inicial 10. O comando **export** faz com que a nova variável seja conhecida por todos os processos a partir deste *shell.* Os nomes das variáveis de ambiente são, tradicionalmente, definidas usando apenas letras maiúsculas. Entretanto, isto não é obrigatório. Você pode também usar letras minúsculas. Mas, **CUIDADO!** O Linux é *case sensitive*. Isto significa que o sistema diferencia letras maiúsculas de letras minúsculas. Portanto, o comando

```
teste=10; export teste
```

cria uma nova variável de ambiente chamada teste e que é diferente da variável TESTE criada anteriormente.

É importante observar que as **variáveis de ambiente** definidas a partir da linha de comando são temporárias. Para criar uma variável permanente, deve-se acrescentar a definição e o comando **export**no arquivo .bashrc (no caso de shell *bash*).

Para excluir uma variável de ambiente, deve-se usar o comando **unset**. Por exemplo, o comando

```
unset teste
```

exclui a variável de ambiente teste criada no exemplo anterior.

Para alterar o valor de uma variável de ambiente, basta fornecer o nome da variável e o novo valor a ser atribuído a variável. Para a variável de ambiente TESTE definida anteriormente

nesta seção, podemos digitar

```
TESTE=200
```

e a variável TESTE passa a ter valor 200. Neste caso, o conteúdo da variável é alterado. Pode ser que ao invés de alterar o conteúdo, você queira apenas acrescentar mais alguma informação ao conteúdo armazenado em uma variável. Por exemplo, suponha que você queira acrescentar o diretório /teste/bin no caminho de busca, ou seja, no PATH. Devemos, então digitar

```
PATH=$PATH:/teste/bin
```

O símbolo **$** usado acima, antes de PATH, informa ao *shell* para usar o conteúdo da variável de ambiente PATH.

Suponha agora que temos “USER=aluno” e que queremos “USER=aluno linux”. Então podemos definir

```
USER=”$USER Linux”
```

As aspas acima são necessárias devido ao espaço em branco existente no novo conteúdo da variável USER. Note que novamente usamos **$** para indicar ao *shell* que se deve substituir o nome USER pelo conteúdo da variável ambiente USER.

Os exemplos mostrados acima são alterações temporárias no ambiente do usuário. Para tornar uma alteração efetiva, deve-se alterar o arquivo de configurações do usuário (.bashrc no *shell bash*). Por exemplo, suponha que queremos personalizar o *prompt* das linhas de comando. Podemos então incluir no arquivo de configurações

```
export PS1=”[\W] “
```

onde o novo *prompt* apenas exibe o nome do diretório atual de trabalho do usuário (\W) entre dois cochetes.

Consulte o **manual on line** para conhecer um pouco mais sobre os comandos embutidos no seu shell (por exemplo, “man bash” para ler sobre o *script bash*).

## Variáveis de ambiente em um programa C

O terceiro argumento da função main() é a lista de variáveis de ambiente. Em um programa C, pode-se usar a função putenv() para criar/alterar variáveis de ambiente, a função getenv() para obter o valor das variáveis e a função unsetenv() para remover variáveis.

O exemplo abaixo mostra a alteração de uma variável de ambiente (LOGNAME) e a inclusão de duas novas variáveis (TESTE1 e TESTE2).

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>
int main(int argc, char * const argv[], char * const envp[])
```

```
{
  printf("*** Valor atual da variavel LONGNAME\n");
  printf("getenv(LOGNAME) = %s;\n\n", getenv("LOGNAME"));
  /* inclui duas novas variáveis de ambiente */
  if (putenv("TESTE1=100") == -1)
    {
      printf("%s: putenv()\n", strerror(errno));
      exit(1);
    }
  if (putenv("TESTE2=teste de variaveis") == -1)
    {
      printf("%s: putenv()\n", strerror(errno));
      exit(2);
    }
  /* altera valor da variável de ambiente LOGNAME */
  if (putenv("LOGNAME=linux") == -1)
    {
      printf("%s: putenv()\n", strerror(errno));
      exit(3);
    }
  /* Verifica o valor de LOGNAME, TESTE1 e TESTE2 */
  printf("*** Valor final das variaveis\n");
  printf("getenv(LOGNAME) = %s;\n", getenv("LOGNAME"));
  printf("getenv(TESTE1)  = %s;\n", getenv("TESTE1"));
  printf("getenv(TESTE2)  = %s;\n", getenv("TESTE2"));
}
```

É possível criar, alterar e deletar variáveis de ambiente, mas as alterações só são válidas durante a execução do programa. Isto significa que as alterações introduzidas pelo programa no ambiente são feitas em outra área de memória e monitoradas pelo sistema enquanto o programa está sendo executado. Não é possível alterar as variáveis de ambiente do *shell* pois as variáveis do *shell* são definidas na própria memória privada do *shell.*

## Alteração do shell

É possível mudar de *shell* em tempo de execução. Para isto, basta digitar o nome do novo *shell* na linha de comandos. Para encerrar o novo *shell* e voltar para o shell anterior basta digitar *exit*.

Para ver quais os *shells* que estão disponíveis no sistema, basta verificar o conteúdo do arquivo */etc/shells*.

Para mudar o *shell* padrão, o usuário pode usar o comando **chsh** (*change shell*). Suponha, por exemplo, que queremos adotar como padrão o shell *ash*, então podemos digitar

```
chsh -s /bin/ash
```

O Linux apenas pedirá a senha do usuário logado para confirmar a alteração. Na próxima vez que o usuário logar no sistema, o *ash* será o seu *shell* padrão. O uso do comando *chsh*, sem

parâmetros, fará com que, além da senha, o Linux também solicite o nome do novo *shell.* Neste caso, deve-se fornecer o caminho completo do novo *shell.*

## Programação em shell script

Como vimos acima, o *shell* funciona como mediador entre o usuário e o **kernel** do sistema. Quando o usuário digita um comando, o *shell* analisa o comando e solicita ao kernel a execução das ações correspondentes ao comando em questão.

Normalmente, o usuário (principalmente se ele é o administrador do sistema) executa um mesmo conjunto de tarefas diariamente. O ideal então é que o usuário possa automatizar as suas tarefas, ou seja, o usuário digita um único comando e o *shell* o interpreta como um conjunto de comandos a serem executados pelo sistema. Este tipo de automatização de tarefas é possível através da programação *shell.*

Abaixo são mostrados alguns exemplos de *scripts*.

**EXEMPLO 1**

Suponha que você queira bloquear o acesso de um usuário ao sistema e caso o usuário tente acessar o sistema mostrar uma mensagem de aviso. Para isto é necessário: criar um *shell script* que será usado na inicialização da conta do usuário e definir este *script* para ser chamado quando o usuário logar na conta. Por exemplo, queremos bloquear a conta do usuário aluno. Então, devemos

**a)** criar o *script* abaixo (sem a numeração do lado esquerdo).

```
1 #!/bin/sh
2 echo '**********************************************'
3 echo '* Sua conta foi encerrada. Procure o suporte. *'
4 echo '**********************************************'
5 sleep 10s
6 exit
```

- A linha 1 indica que o script deve ser executado pelo shell *sh*. O símbolo # significa início de um comentário (o resto da linha não é interpretada), mas quando **#!** aparecem na primeira linha de um *script*,estamos informando ao sistema para procurar o programa definido a seguir (neste caso o /bin/sh) e transmitir o resto do arquivo a esse programa.
- O comando echo (linhas 2 a 4) imprime o texto entre aspa na tela. O comando "sleep 10s" (linha 5) dá uma pausa de 10 segundos antes de continuar a execução do *script*.
- O comando exit (linha 6) finaliza o *script*.

Salve o script acima em /bin com o nome nsh e torne este arquivo executável (use o comando "chmod +x nsh").

**b)** alterar o arquivo /etc/passwd para que o novo *script* seja chamado quando o usuário logar no sistema.

```
aluno:x:501:501::/home/aluno:/bin/nsh
```

Agora, quando o usuário aluno tentar logar no sistema, ele receberá a mensagem que a conta foi encerrada.

**EXEMPLO 2**

Abaixo temos um segundo exemplo de *script* de shell. Este *script* mostra quais as **permissões de acesso** do usuário em relação a um determinado arquivo.

```
1. #!/bin/bash
2. echo -n 'Forneça o nome do arquivo a ser verificado: '
3. read
4. if [ $REPLY ] #usuário digitou o nome do arquivo?
5. then
6.     arq=$REPLY
7.     if [ -e $arq ]; then # o arquivo existe ?
8.        echo 'o arquivo existe'
9.        if [ -r $arq ]; then # o usuário pode ler o arquivo?
10.          echo 'você pode ler o arquivo'
11.       else
12.          echo 'você não pode ler o arquivo'
13.       fi
14.       if [ -w $arq ]; then # o usuário pode alterar o arquivo?
15.          echo 'você pode alterar o arquivo'
16.       else
17.          echo 'você não pode alterar o arquivo'
18.       fi
19.       if [ -x $arq ]; then # o usuário pode executar o arquivo?
20.          echo 'você pode executar o arquivo'
21.       else
22.          echo 'você não pode executar o arquivo'
23.       fi
24.       if [ -d $arq ]; then # o arquivo é um diretório?
25.          echo 'O arquivo é um diretório'
26.    fi
27.    else
28.       echo 'o arquivo nâo existe'
29. fi
30. else
31.    echo 'Você não forneceu o nome do arquivo'
32. fi
33. exit
```

Em relação ao *script* acima podemos comentar:

- A linha 1 é um comentário e especifica que o *script* deve ser executado pelo shell *sh*.
- A linha 2 exibe na tela a mensagem 'Forneça o nome do arquivo a ser verificado: '. O parâmetro **-n** do comando *echo* indica que o cursor não deve mudar de linha após a exibição da frase. Em relação ao comando *echo,* é também importante observar que podemos usar tanto aspas simples quanto aspas duplas, embora sejam interpretadas de forma diferente pelo *shell.* Por exemplo, suponha uma variável denominada TESTE e que tenha armazenado o valor Maria. A execução dos comandos

```
echo 'TESTE = $TESTE'
echo "TESTE = $TESTE"
```

mostram, respectivamente, os seguintes resultados

```
TESTE = $TESTE
TESTE = Maria
```

No primeiro caso, o uso de aspas simples informa ao *bash* para imprimir o conteúdo do *string*, sem nenhuma preocupação adicional. No segundo caso, o uso de aspas duplas faz com que o *sh* substitua o nome da variável TESTE pelo seu conteúdo. O símbolo **$** indica ao *sh* quem é variável dentro do string.

- O comando *read* da linha 3 recebe o nome do arquivo digitado pelo usuário e o armazena na variável padrão REPLY. Pode-se também usar uma variável qualquer para receber o nome do arquivo. Por exemplo, você pode alterar a linha 3 para “read arq”. Neste caso, o *sh* passa a executar duas ações quando interpreta a linha 3: primeiro, cria a variável arq, e segundo, armazena o nome do arquivo nesta variável.
- A linha 4 do *script* verifica se o usuário digitou algo (ele pode ter apenas teclando ENTER). O comando *if* possui a seguinte estrutura

```
if [ condição ]
then
    comandos
else
    comandos
fi
```

Antes de discutirmos o *script*, algumas observações em relação ao comando *if* tornam-se necessárias. A condição (ou condições) a ser testada deve ser colocada entre colchetes (pode-se também usar explicitamente a palavra *test* no lugar dos colchetes, por exemplo, você pode substituir *if [ $REPLY ]* por *if test $REPLY*). Além disso, deve existir um espaço entre a condição e o colchete (abrindo e/ou fechando). O *sh* retorna 0 ou 1 como resultado do teste, dependendo se a condição é verdadeira (valor zero) ou falsa (valor 1). Caso a condição seja verdadeira, são executados os comandos definidos logo após o comando *then*. Caso a condição seja falsa, são executados os comandos logo após o comando *else* (você não é obrigado a definir um *else* para cada *if*). O comando *if* é fechado com um comando *fi*.

Olhando novamente o segundo exemplo de *script* podemos notar a seguinte estrutura

```
4 if [ $REPLY ] #usuário digitou o nome do arquivo?
5 then
     executa linhas de 6 a 29
30 else
     executa linha 31
32 fi
```

A linha 4 verifica se a variável REPLY tem algum valor armazenado. Caso o resultado do teste seja verdadeiro, são executados os comandos da linha 6 a linha 29; caso o resultado do teste seja falso, apenas a linha 32 é executada.

- A linha 6 define a variável arq e copia o conteúdo da variável REPLY para a nova variável. Podemos aqui fazer três observações: primeiro, não existe espaço em branco entre as variáveis e o símbolo de atribuição (“=”); segundo, a criação da variável arq não é necessária, poderíamos continuar usando a variável REPLY no resto do *script*; e terceiro, a variável arq é uma variável local (não existe um comando **export** para esta variável), isto significa que esta variável existe somente durante a execução do *script*.
- A linha 7 mostra o comando *if* com o operador **-e**. O uso deste operador faz com que o *sh* verifique a existência do arquivo cujo nome foi informado pelo usuário. Podemos ver

que o *script* tem a seguinte estrutura a partir deste teste

```
7 if [ -e $arq ]; then # o arquivo existe ?
        executa linhas de 8 a 26
27 else
        executa linha 28
29 fi
```

Note que temos dois comandos na linha 7: o comando *if* e o comando *then*. Quando mais de um comando são colocados em uma mesma linha, eles devem ser separados por um ponto-e-vírgula.

- As linhas de 8 a 26 utilizam operadores junto com o comando *if* para verificar as **permissões** do arquivo. Abaixo mostramos alguns operadores que podem ser usados para testar arquivos (digite “man test” para obter mais informações).

**-b** : o arquivo existe e é um arquivo é especial de bloco ?
**-c** : o arquivo existe e é um arquivo é especial de caractere ?
**-d** : o arquivo existe e é um diretório ?
**-e** : o arquivo existe ?
**-f** : o arquivo existe e é um arquivo normal ?
**-g** : o arquivo possui permissão especial SGID ?
**-k** : o arquivo possui permissão especial sticky bit ?
**-r** : o arquivo existe e o usuário pode lê-lo ?
**-s** : o arquivo existe e tem tamanho maior que zero ?
**-u** : o arquivo possui permissão especial SUID ?
**-x** : o arquivo existe e o usuário pode executá-lo ?
**-w** : o arquivo existe e o usuário pode alterá-lo ?
**-G** : o arquivo existe e pertence ao grupo do usuário ?
**-L** : o arquivo existe e é um link simbólico ?
**-O** : o arquivo existe e o usuário é dono do arquivo ?
**arq1 nt arq2** : o arquivo arq1 é mais novo que o arquivo arq2 ?
**arq1 ot arq2** : o arquivo arq1 é mais antigo que o arquivo arq2 ?

**EXEMPLO 3**

O terceiro *script* é uma ferramenta de *backup* para arquivos com extensão *txt*. O usuário fornece o nome do diretório e todos os arquivos .txt deste diretório são copiados para o diretório *backup*. Se o diretório *backup* já existe, ele é inicialmente apagado e depois criado.

```
1. #!/bin/sh
2. if [ ! $1 ]
3. then
4.     echo ‘Você deve fornecer o nome do diretório’
5.     exit 1
6. fi
7. if [ ! -d $1 ]
8. then
9.     echo “$1 não é um diretório”
10.    exit 1
11. fi
12. rm -fr ~/backup
13. mkdir ~/backup
14. for i in $1/*; do
15.     echo $i
```

```
16.     cp -f $i ~/backup/
17. done
18. exit
```

Podemos comentar em relação ao *script* acima:

- A linha 1 especifica que o *script* deve ser executado pelo shell *sh*.
- A linha 2 testa se o usuário forneceu algum argumento de linha de comando. A variável **$1** corresponde ao primeiro argumento, a variável **$2** corresponde ao segundo argumento, e assim por diante. A variável **$0** possui o nome do programa e a variável **$*** possui a lista das variáveis (**$0**, **$1**, **$2**, …). O símbolo **!**é o símbolo de negação (NÃO), portanto estamos perguntando na linha 2 se o agumento não foi fornecido pelo usuário. Caso seja verdade, o programa mostra a frase 'Você deve fornecer o nome do diretório' e encerra a execução de forma anormal (*exit 1*).
- A linha 7 verifica se o nome fornecido pelo usuário não é o nome de um diretório. Caso isto seja verdade, o programa é encerrado.
- A linha 12 apaga o diretório *~/backup* e todos os seus arquivos, caso este diretório exista.
- A linha 13 cria o diretório *~/backup*. Note que o diretório é filho do diretório principal (raiz) do usuário.
- Nas linhas de 14 a 17 temos a cópia dos arquivos. A linha 14 define que para cada arquivo do diretório fornecido pelo usuário, deve-se executar os comandos das linhas 15 (exibe o nome do arquivo na tela) e 16 (copia o arquivo para diretório *~/backup*). As instruções dentro do laço *for* serão executadas tantas vezes quantas forem o número de arquivos com extensão txt no diretório fornecido pelo usuário. Portanto, a variável local armazena um nome do arquivo diferente a cada execução do laço.
- A linha 18 encerra o *script*.

## Operações aritméticas

Existem várias formas de executar uma operação aritmética no shell.

- Pode-se usar o comando **expr** para operações com números inteiros como mostrado abaixo.

```
[~] expr 5 + 2
7
[~] expr 5 – 2
3
[~] expr 5 \* 2
10
[~] expr 5 / 2
2
[~] expr 5 % 2
1
```

Portanto, o comando reconhece os cinco operadores aritméticos mostrados na tabela abaixo.

| Operador | Significado |
|---|---|
| + | Soma |
| – | Subtração |
| * | Multiplicação |

| | |
|---|---|
| / | Divisão |
| % | Resto da divisão |

Note que os operandos e os operadores precisam ser separados por espaço. Além disso, o operador * (multiplicação) precisa ser antecedido pela barra invertida para que não seja interpretado pelo shell.

- Para operações aritméticas com ponto flutuante, deve-se usar a linguagem **bc**. Nos exemplos abaixo, pede-se que o resultado seja apresentado com até duas casas decimais.

```
[~] bc <<< “scale=2;5.3+2.1”
7.4
[~] bc <<< “scale=2;5.3-2.1”
3.2
[~] bc <<< “scale=2;5.3*2.1”
11.13
[~] bc <<< “scale=2;5.3/2.1”
2.52
[~] bc <<< “scale=2;5.3%2.1”
.008
```

A linguagem **bc** é muito rica permitindo resolver expressões aritméticas mais complexas que as cinco operações mostradas acima.

- Pode-se também usar o interpretador aritmético $ (ele não está disponível em todos os shells). Este interpretador só trabalha com números inteiros.

```
[~] echo $((5 + 2))
7
[~] echo $((5 – 2))
3
[~] echo $((5 * 2))
10
[~] echo $((5 / 2))
2
[~] echo $((5 % 2))
1
```

Note que é necessário um duplo parêntesis com o interpretador $. Os parêntesis mais internos engloba a expressão a ser calculada, enquanto os parêntesis mais externos delimita o escopo do $.

## Mais exemplos

- O exemplo abaixo mostra a implementação de uma calculadora com as quatro operações. Note que três linhas do programa possuem mais de um comando e que eles são separados por “;”. Além disso, a última linha chama **bc** (*basic calculator*) para resolver a operação e fornecer o resultado com até duas casas decimais (as operações feitas pelo *bash* não trabalham com ponto flutuante).

```
#!/bin/bash
format_num='^[0.0-9.0]+$'
format_op='^[-,+,/,*]$'

echo -n "escolha um numero: "; read numero1
if ! [[ $numero1 =~ $format_num ]] ; then
echo "valor nao numerico"; exit 1
fi

echo -n "escolha a operacao '- + * /': "; read func
if ! [[ $func =~ $format_op ]] ; then
echo "operacao nao valida"; exit 1
fi

echo -n "escolha outro numero: "; read numero2
if ! [[ $numero2 =~ $format_num ]] ; then
echo "valor nao numerico"; exit 1
fi

echo -n "Resultado = "; echo "scale=2; $numero1$func$numero2" | bc
```

- Outra forma de implementar uma calculadora é mostrada abaixo.

```
#!/bin/bash
echo -n "escolha um numero: "; read numero1
echo -n "escolha outro numero: "; read numero2

let soma=$numero1+$numero2
echo "$numero1 + $numero2 =" $soma

let subtracao=$numero1-$numero2
echo "$numero1 – $numero2 =" $subtracao

let multiplicacao=$numero1*$numero2
echo "$numero1 * $numero2 =" $multiplicacao

let divisao=$numero1/$numero2
echo "$numero1 / $numero2 =" $divisao

let resto=$numero1%$numero2
echo "$numero1 % $numero2 =" $resto
```

- Este exemplo ler e exibe as linhas de um arquivo. O nome do arquivo de entrada deve ser fornecido na linha de comandos.

```
#!/bin/bash
arquivo=$1
while read -r linha
do
    echo $linha
done < $arquivo
```

- Para ler os campos de */etc/passwd* que são separados por ":", basta implementar a versão abaixo.

```
#!/bin/bash
i=0
while IFS=: read user pass uid gid full home shell
do
    let i=$i+1
    echo "*$i* User = $user, UID = $uid, GID = $gid, Home = $home, Shell = $shell"
done < /etc/passwd
```

- O exemplo abaixo usa *printf* para formatar a saída.

```
#!/bin/bash
i=0
while IFS=: read user pass uid gid full home shell
do
    let i=$i+1
    printf "*%2d* User = %s, UID = %d, GID = %d, Home = %s, Shell = %s\n" $i $user
$uid $gid $home $shell
done < /etc/passwd
```

Para gravar a saída em um arquivo, basta alterar o final da linha *printf* para

```
printf "*%2d* User = %s, UID = %d, GID = %d, Home = %s, Shell = %s\n" $i $user $uid
$gid $home $shell >> $1
```

onde $1 é um nome de arquivo fornecido na linha de comandos.

- É muito simples definir e acessar vetores no *bash*. No exemplo abaixo, dois parâmetros especiais são usados: # calcula o número de posições do vetor e @ expande todas as ocorrências do vetor.

```
#!/bin/bash
Vetor=( 'Debian Linux' 'Redhat Linux' Ubuntu Linux )
num=${#Vetor[@]}
for ((i = 0; i < $num; i++))
do
    echo ${Vetor[$i]}
done
```

- O exemplo abaixo mostra um menu com três opções de comandos.

```
#!/bin/bash
selection=
until [ "$selection" = "0" ]; do
echo ""
echo "Escolha o comando"
echo "1 – Exibe arquivos/diretorios"
echo "2 – Mostra diretorio de trabalho"
echo "3 – Uso do disco"
echo ""
echo "0 – Encerra programa"
echo ""
```

```
echo -n “Opcao selecionada: ”
read selection
echo “”
case $selection in
1 ) ls ;;
2 ) pwd ;;
3 ) df ;;
0 ) exit ;;
* ) echo “Escolha 1, 2, 3 ou 0”
esac
done
```

## Observações

- Consulte a página do grupo **SS64** para mais informações sobre os comandos e as opções do *shell bash* ou digite

  ```
  man bash
  ```

  na linha de comandos do shell.

- Existe outra forma de executar os scripts. Ao invés de alterar as **permissões de acesso** do script com o comando **chmod**, pode-se simplesmente especificar o nome do shell na linha de comandos. Por exemplo, suponha que queremos executar o arquivo *teste* usando o *script bash*. Então podemos digitar

  ```
  bash teste
  ```

- Para alterar a saída e/ou a entrada dos comandos shell, use **redirecionadores**.