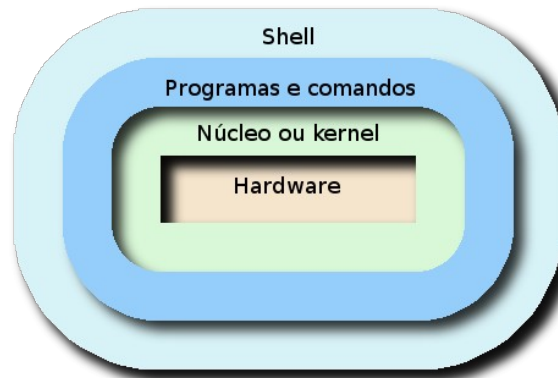


SHELL SCRIPT



Arquitetura

Ordem de execução:

1. Resolução de redirecionamentos
2. Substituição de variáveis
3. Substituição de meta caracteres
4. Passagem do comando para o Kernel

Variáveis e variáveis de ambiente:

A declaração de variáveis não é necessária. No Shell, as variáveis são criadas no momento que são usadas a primeira vez. A tipagem é automática e de acordo com o conteúdo. Ou seja, caso você tente somar um inteiro com uma string você receberá um erro como retorno.

Atribuindo valor a uma variável:

```
var1="String"  
var2="1"
```

Os vetores tem o mesmo conceito das variáveis. Assim, para criar um vetor basta adicionar o índice ao nome da variável:

```
meuvetor[0]="String"  
meuvetor[1]="10"
```

Observe que na atribuição não é necessário o uso do cifrão (\$). Porém, o cifrão é obrigatório quando a variável for referenciada:

```
echo $var1  
echo $var2  
echo ${meuvetor[0]}  
echo ${meuvetor[*]}
```

NOTA: As variáveis podem ser referenciadas com as sintaxes `$var1` ou `${var1}`. Ambas tem o mesmo efeito. A diferença é que a segunda delimita o nome da variável. Isso pode ser útil quando a variável estiver referenciada no meio de um texto. No caso de vetores, é obrigatório o uso das chaves para referenciá-los.

O Bash possui variáveis de ambiente que podem ser usadas por qualquer script. Você pode visualizar todas elas com o comando `env`. Alguns exemplos:

`$?` – Contém o código de retorno do último comando executado.
`$PWD` – Contém o caminho absoluto do diretório atual.
`$PATH` – Contém os diretórios onde o Bash irá procurar por binários.
`$RANDOM` – É uma variável especial para geração de números aleatórios entre 0 e 65535.

Parâmetros:

A passagem de parâmetros no Bash é um recurso que pode e deve ser usado para definir o comportamento de um script. Cada parâmetro deve ser passado separado por um espaço. Eles serão identificados dentro do script por variáveis especiais reservadas.

O primeiro parâmetro passado será recebido dentro do script pela variável `$1`, o segundo pela `$2` e assim

sucessivamente. A partir da `{10}` é obrigatório o uso das chaves para desambiguação.

Sintaxe:

```
./meuscript.sh parametro1 parametro2 parametro2
```

Exemplo:

```
./meuscript.sh start
```

Além das variáveis que receberão os parâmetros, ainda há outras relacionadas a passagem de parâmetros que podem ser usadas pelo script:

`$0` – Variável que recebe o comando executado.

`$#` – Variável que contém a quantidade de parâmetros passados.

`$*` ou `$@` – Mostra todos os parâmetros passados.

Condicional **if**:

O *if* testa se o comando foi ou não executado com sucesso, não se preocupando com o que o comando envia para a saída padrão ou para a saída de erro padrão. Para isso, ele faz uso do código de retorno (`$?`) da execução do comando.

Sintaxe:

```
if <comando>
then
    <comando>
eles
    <comando>
fi
```

Exemplo:

```
if ls /etc
then
    echo "Comando ls executado com sucesso."
eles
    echo "Erro executando comando ls."
fi
```

Para fazer testes/comparação de variáveis, usar operadores lógicos ou testes em arquivos é necessário usar o *if* associado a um comando específico. O comando *test* representa a melhor opção para esse tipo de necessidade.

Exemplo:

```
if test $var1 -eq 0 -a $var2 == "curso"
then
    echo "Todas as condições aceitas."
eles
    echo "Erro em pelo menos uma das condições"
fi
```

No exemplo acima, o comando *test* verifica se a variável `$var1` está preenchida com o número inteiro `0` e (and) a variável `$var2` está preenchida com a string `curso`. Mais na frente você encontrará tabelas com todas as opções aceitas pelo comando *test*.

Condicional **case**:

Diferente do *if*, o *case* testa a coincidência de valores e não o resultado de comandos. Ele é bastante útil quando temos várias possibilidades.

Sintaxe:

```
var="valor2"
case $var in
    valor1)
        <comando>
        <comando>
;;
```

```

        valor2)
            <comando>
        ;;
        valor3)
            <comando>
        ;;
        *)
            <comando>
        ;;
    esac

```

Uma aplicação bastante frequente do *case* é associado ao recebimento de parâmetros.
Exemplo:

```

case $1 in
    start)
        /etc/init.d/meuservico start
        echo "Serviço inicializado com sucesso."
    ;;
    stop)
        /etc/init.d/meuservico stop
        echo "Serviço parado com sucesso."
    ;;
    valor3)
        /etc/init.d/meuservico stop
        echo "Serviço parado com sucesso."
        /etc/init.d/meuservico start
        echo "Serviço inicializado com sucesso."
    ;;
    *)
        echo "Parâmetro não reconhecido."
        echo "Use: $0 { start | stop | restart }"
    ;;
esac

```

No exemplo acima, o script deverá receber um parâmetro, que será testado e definirá o comando que será executado. No caso do parâmetro não ser informado ou não ser reconhecido, uma mensagem de erro alertará o usuário.

Laço **for**:

O *for* é o gerador de laços limitados do Bash. No Shell, seu funcionamento é bem mais poderoso que em várias outras linguagens.

Sintaxe:

```

for var in valor1 valor2 valor3 valor4
do
    <comando>
    <comando>
done

```

A cada loop, a variável *var* assume um valor da lista *valor1 valor2 valor3 valor4*.

Exemplo:

```

for var in $(seq 10)
do
    echo "O valor de var agora é: $var"
done

```

No exemplo acima, usamos o *\$(seq 10)* para dizer ao Shell que execute primeiro o comando *seq 10*, que irá criar uma lista de números de 1 até 10. Só então o *for* será executado, percorrendo toda a lista e executando o comando *echo* em cada loop.

Dessa forma, qualquer comando pode ser executado para gerar a lista que será usada pelo *for*. Mas atenção, o *for* usa a variável de ambiente *IFS* para delimitar os campos da lista. Por padrão, o *IFS* contém “espaço”, “TAB” e “enter”. Qualquer um deles que ocorra na lista será considerado pelo *for* o limite de um campo.

Laço while:

O *while* é um gerador de laço que testa comandos e não valores. Assim como o *if*, o *while* precisa da ajuda de comandos como o *test* para testar valores, variáveis ou arquivos.

Sintaxe:

```
while <comando>
do
    <comando>
    <comando>
done
```

Exemplo:

```
var1=1
while test $var1 -lt 10
do
    echo "Variável var1 agora é $var1"
    var1=$((var1+1))
done
```

No exemplo acima, usamos o comando *test* em cada loop para testar se a condição ainda é verdadeira. A cada loop usamos o operador aritmético $(())$ para incrementar o valor da variável *\$var1*.

Opções do comando test:

Opções para arquivos	
Opção	Verifica se
-x arq	arq existe e possui direito de execução
-e arq	arq existe
-s arq	arq existe e é maior que zero
-f arq	arq existe e é um arquivo regular
-d arq	arq existe e é um diretório
-r arq	arq existe e possui permissão de leitura
-w arq	arq existe e possui permissão de escrita

Opções para strings	
Opção	Verifica se
s1 == s2	String s1 e s2 são iguais
s1 != s2	String s1 e s2 são diferentes
-z string	Tamanho da string é zero
-n string	Tamanho da string é maior que zero
string	Tamanho da string é maior que zero

Opções para números	
Opção	Verifica se
n1 -eq n2	n1 é igual a n2
n1 -ne n2	n1 não é igual a n2
n1 -gt n2	n1 é maior que n2
n1 -ge n2	n1 é maior ou igual a n2
n1 -lt n2	n1 é menor que n2
n1 -le n2	n1 é menor ou igual a n2

Operadores	
-o	Ou (or)
-a	E (and)
()	Agrupar
!	Negar