Programação SHELL SCRIPT

Instrutor – Airton Kuada
email - airton@fesppr.br
Curitiba, julho de 2004

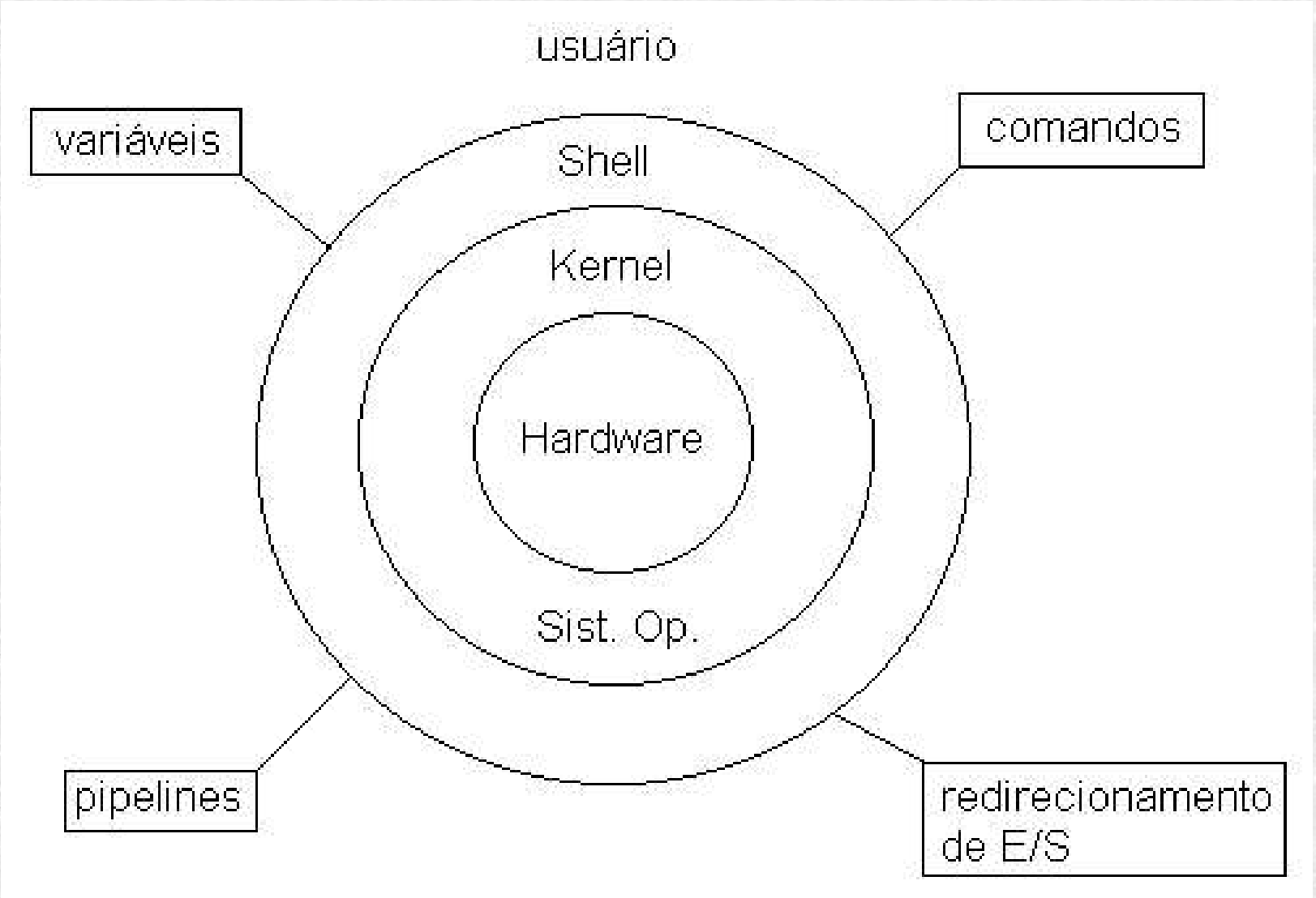
Agenda

- Introdução
- Funções de Terminal
- Estrutura de um Shell Script
- Apresentação em mensagens
- Variáveis
- Comando de controle de fluxo
- Funções

Introdução

- Finalidade.
  - Faz a interação entre o usuário e o Sistema Operacional
- Funcionalidade do Shell
  - Através de linha de comando
    - Pesquisa por um comando e executa o programa a ele associado.
    - Subistitui os valores da variável Shell por variáveis associadas
    - Manipula redirecionalmento de E/S e Pipelines
- Contém interface de programação interpretada

Introdução


usuário
variáveis
comandos
Shell
Kernel
Hardware
Sist. Op.
pipelines
redirecionamento
de E/S

Tipos de SHELL

- Shells mais utilizados
  - bash - LINUX SHELL
  - sh - POSIX SHELL
  - ksh - KORN SHELL
  - bsh - BOURN SHELL
  - csh - C SHELL
  - rksh - KORN SHELL RESTRITO
  - rbsh - BOURN SHELL RESTRITO

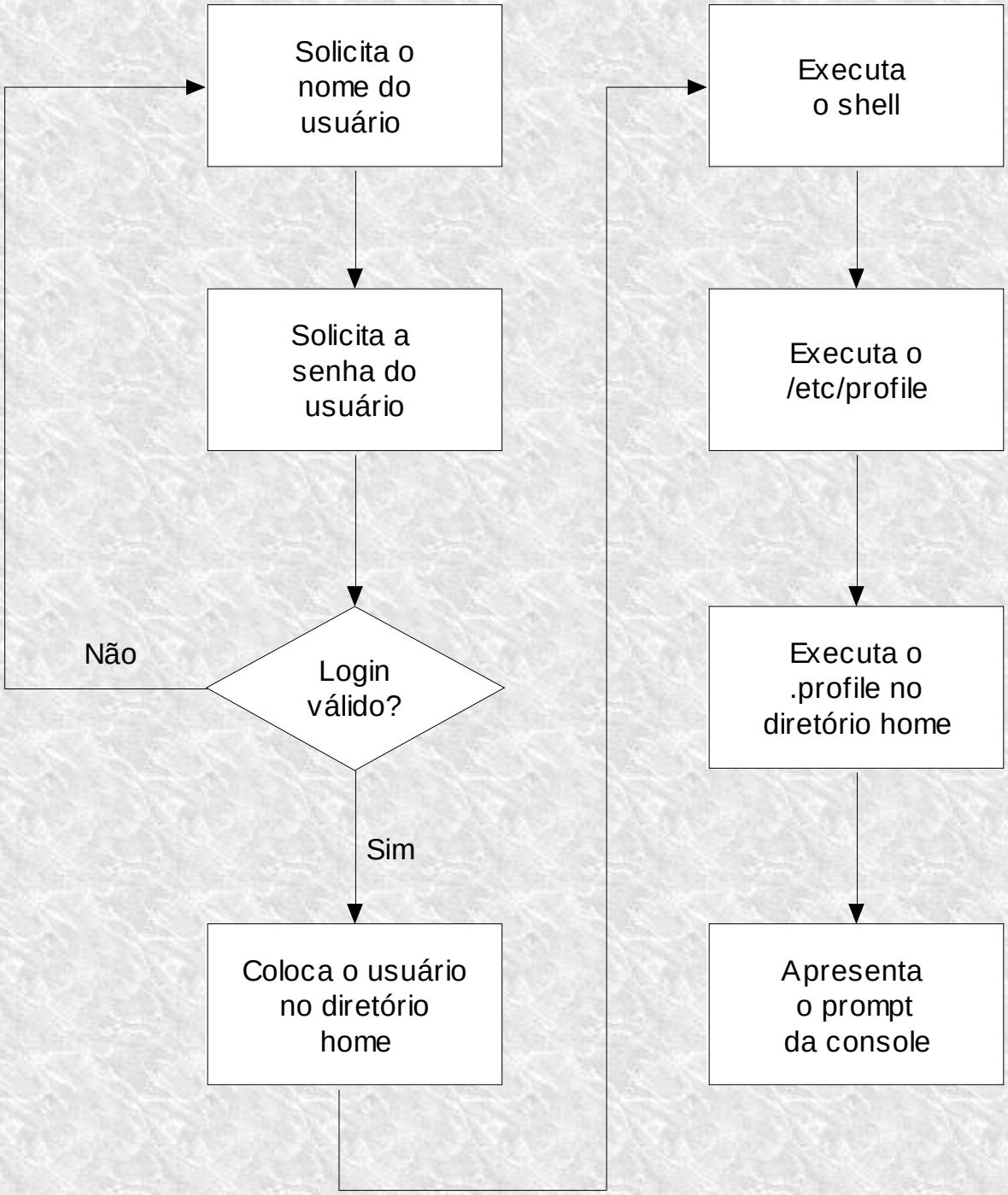
Ambiente do Usuário

- É o ambiente criado para o usuário, após o momento que é aceito o seu pedido de sessão com o Sistema Operacional (Login)
- Seu ambiente contém as seguintes informações:
  - Nome do Usuário
  - Identificaçao do usuário
  - Diretório Home
  - Caixa postal
  - Path
  - Qual o Shell que está sendo utilizado
  - Tipo de Terminal

Logon no Sistema

- Tela de Logon do Sistema
- Usuário informa UserId e Senha
- Sistema Operacional faz autenticação (/etc/passwd)
- Usuário é levado até seu diretório HOME
- Shell é iniciado
- Profile do Sistema Operacional é executado
- Profile do Usuário é executado
- Apresentação do Prompt de Sessão (# ou $).

Logon no Sistema



Shell Script

- Shell Script é uma sequência de comandos que são armazenados em um arquivo do tipo texto e são executados sequencialmente pelo interpretador de comando (Shell) . Esta sequencia de comandos pode ser executado diretamente na console do computador.
- Podemos dizer um Shell Script é uma automatização de uma sequencia comandos que são executados repetidamente pelo usuário de um computador.

Shell Script

- Por que utilizar Shell Script
  - Economizar tempo
  - Automatizar tarefas do dia a dia
  - Parte da administração pode ser automatizada
  - Podemos criar novos comandos
  - Desenvolvimento rápido
  - Facilidade de desenvolvimento e manutenção

Shell Script - Exemplo

```
#!/bin/bash
#-----------------------------------------------------------#
#       Este é um exemplo de Shell Script                   #
#       Objetivo – Mostrar os arquivos do                   #
#                  diretório corrente, utilizando           #
#                  paginação                                #
#-----------------------------------------------------------#
clear
echo "Os arquivos deste diretórios são: "
ls -la | more

#-------- Fim do Script -----------------------------------#
```

Shell Script - Estrutura

```
#!/bin/bash        (obrigatório sempre)
Comando1
comando2
comando3
comando4
# Esta linha é um comentário e não será executado
comando5
comando6
```

Shell Script

- Escrever o Shell Script
- Executar o Script
  - . arquivo_do_script
  - bash arquivo_do_script
- Atribuir a propriedade de execução no arquivo
  - chmod +x arquivo_do_script ou
  - chmod 755 arquivo_do_script
- Para ativar o modo debug no Script é necessário acrescentar o parametro -x como mostrado abaixo.
  - #!/bin/bash -x

Metacaracteres

Metacaracter	O que faz
*	Substitui qualquer string ou grupo de caracteres
?	Substitui um único caracter
[....]	Substitui qualquer dos caracteres contidos entre os colchetes

Exemplo	Significado
ls *	Mostrará todos os nomes de arquivos
ls a*	Mostrará todos os nomes de arquivos que iniciam com a letra **a**
ls *.c	Mostrará todos os nomes de arquivos que possuem a extensão **.c**
ls ut*.c	Mostrará todos os nomes de arquivos que iniciam com as letras **ut** e possuem a extensão **.c**
ls ?	Mostrará todos os nomes de arquivos que possuem 1 caracter.
ls fo?	Mostrará todos os nomes de arquivos com três letras e que iniciam com as letras **fo**.
ls [abc]*	Mostrará todos os nomes de arquivos que iniciam a ou b ou c.
ls [a-d]*	Mostrará todos os nomes de arquivos que iniciam a, b, c ou d.

Execução de Comandos

- Normalmente executamos apenas um comando por vez em um prompt de comando.
- Exemplo
  - $ ls
  - $ date
  - $ who
- O Unix permite que o usuário execute dois ou mais comandos, utilizando o mesmo prompt.
- Exemplo
  - $ ls;date;who

Redirecionamento de E/S

- Os periféricos de entrada e saída no mundo Unix estão associados a arquivos que estão localizados no diretório /dev. Ex: console, hda, ttyS*, video, audio
- Para acessar um arquivo, o sistema operacional associa um identificador numérico (handle) para controlar o acesso
- Os handles 0, 1, 2 estão constantemente ligados aos periféricos teclado, monitor e monitor respectivamente.

Redirecionamento de E/S

- Aplicações enviam a saída para o terminal de vídeo, exemplo: ls, banner, who
- Aplicações obtém a entrada através do teclado, exemplo: mail, write, cat.
- O Unix permite que seja realizado o redirecionamento da saída, isto é, a saída de uma aplicação que normalmente é a unidade de vídeo (/dev/video) pode ser redirecionada para um arquivo.
- O Unix permite que seja realizado o redirecionamento de entrada, isto é, a entrada de uma aplicação que normalmente é o teclado (/dev/console) pode ser redirecionada para o conteúdo de um arquivo.

Redirecionamento de E/S

- A unidade de vídeo é também conhecida como **saída padrão ou stdout (standard output), handle 1**
- O teclado é também conhecido como **entrada padrão ou stdin (standard input), handle 0**
- Todas mensagens de erros geradas durante a execução de um programa são enviadas para o **handle 2** que também está associado a unidade de vídeo e também é conhecido como **saída de erro padrão ou stderr**
- Quando não é necessário apresentar nenhuma saída, podemos redirecionar a **stdout** e **stderr** para o **BURACO NEGRO** (/dev/null)

Redirecionamento de E/S

- O simbolo de redirecionamento de saída é > e de entrada é <
- Redirecionamento de saída, exemplo:
  - ls -la > saida.txt
- No exemplo acima o resultado do comando ls será enviado o arquivo saida.txt, se por ventura o arquivo já existir, todo o seu conteúdo será apagado.
- Redirecionamento de entrada, exemplo:
  - cat < saida.txt
- No exemplo acima, o comando cat irá mostrar o contéudo do arquivo saida.

Redirecionamento de E/S

- Para adicionar informações a um arquivo, utilizamos o simbolo >>
- Exemplo:
  - ls *.txt > saida.txt
  - ls *.tst >> saida.txt
- No primeiro comando o arquivo saida.txt é inicializado e no segundo comando, o arquivo é incrementado com novas informações
- Podemos iniciar um arquivo também da seguinte forma:
  - > saida.txt

Redirecionamento de E/S

- Podemos redirecionar os erros que podem eventualmente ocorrer durante o processamento de um comando
- Exemplo
  - ls -la *.wxs
  - Se não existir nenhum arquivo com esta extensão então teremos a mensagem de erro abaixo que aparecerá no vídeo
  - /usr/bin/ls: *.wxs: No such file or directory
- Para evitar a situação acima, podemos redirecionar a saida de erro padrão (stderr) para um arquivo
- Exemplo
  - ls -la *.wxs 2> erro.txt

Exercício 1

- Utilizando o comando find, buscar os arquivos que foram alterado hoje e redirecionando a saída para o arquivo **hoje.txt**
- Utilizando o comando find, buscar os arquivo que possuam as extensão .cfg, redirecionando a saída para o arquivo **cfg.txt**
- Juntar o arquivo hoje.txt e cfg.txt em um único arquivo chamado **total.txt**

PIPELINES

- Um PIPE nada mais é do que um local temporario de armazenamento de dados que será repassado para um próximo comando.
- A utilização de PIPES permite conectar a saída de um comando com a entrada de outro, para que ela possa ser tratada por este seguinte.
- O sinal de PIPE é a barra “|”
- Para que um comando possa ter um sinal de PIPE á esquerda, ele deve ler a entrada padrão
- Para que um comando possa ter um sinal de PIPE á direita, ele deve escrever na saida padrão.

PIPELINE

- Exemplo de utilização:
  - Queremos saber quantos arquivos existem em uma determinada pasta.
  - Primeira Forma – Sequencial
    - ls -la > saida.txt
    - wc -l saida.txt
  - Segunda Forma - Pipeline
    - ls -la | wc -l
- No exemplo acima, o comando **ls** é executado integralmete. O resultado é passado para o comando **wc** que irá contar a quantidade de linhas que resultaram do processamento do comando ls.

Exercícios 2

- Mostrar todos os arquivos de um diretório (ls), realizando a paginação (more).
- Mostrar todos os usuários que estão em sessão no sistema (“logados”) (who) em ordem alfabética (sort).
- Mostrar o número de usuários que estão em sessão no sistema.
- Mostrar (cat) todos os usuários cadastrados no sistema, (/etc/passwd) ordenados em ordem alfabética.
- Montar um arquivo (cadastro.txt) com todos os usuários cadastrados no sistema (/etc/passwd) ordenados em ordem alfabética.
- Pesquisar se o usuário “raul” está “logado”. (who+grep)

Filtros

- A maioria dos comandos utilizam apenas a saída padrão (stdout), ou a entrada padrão (stdin)
- Filtros são comandos que laceitam a entrada pela entrada padrão e produz um resultado na saída padrão.
- Exemplo
  - cat < arquivo.txt | sort | uniq > u_name.txt
- No exemplo acima, o comando cat irá mostrar o conteúdo do arquivo. Este conteúdo será ordenado e finalmente o comando uniq irá remover a entrada duplicadas do arquivo.

Apresentação de Mensagens

- Para interagir com o usuário o shell necessita enviar mensagens e mostrar valores de variáveis para a console de vídeo. Para isso é utilizado o comando "**echo**"
- Exemplo: echo [options] [string variaveis ...]
  - echo "Mensagem do Shell"
  - echo 'Mensagem do Shell'
  - echo Mensagem do Shell
- Existem diferenças entre os tres exemplos. Estas diferenças serão abordados adiante.

Echo - Opções

- -n Do not output the trailing new line.
- -e Enable interpretation of the following backslash escaped characters in the strings:
- \a alert (bell)
- \b backspace
- \c suppress trailing new line
- \n new line
- \r carriage return
- \t horizontal tab
- \\ backslash
- Exemplo:
  - echo -e "Amanhã vou pescar \a\t\t com meu amigo\n"

Variáveis

- O Shell permite a criação de variáveis
- Às variáveis são atribuídas valores que são cadeias de caracteres
- Não existe a definição de tipo de variável
- O nomes de variáveis devem obrigatoriamente iniciar com uma letra ou sublinhado podem ser formadas por:
  - Letras
  - Números
  - Sublinhados
- Exemplo: um, var, VAR, VAR1, VAR_1, _var. Uma variável é criada no momento que é realizado uma atribuição.

Variáveis

- Criação e atribuição de valores
  - var=”esta é uma string”
  - var=estaeumastring
  - var=1
- Um detalhe importante: **NÃO PODE EXISTIR ESPAÇOS ENTRE OS ELEMENTOS DA DECLARAÇÃO/ATRIBUIÇÃO**
- Quando a string atribuída a variável possuir espaços, é necessário a utilização de aspas duplas no início e no final da string.
- As variáveis são case-sensitive, isto é, existe diferenças entre maiusculas e minusculas.
- Exemplo: var, Var, vaR, vAr.

Variáveis

- Para apresentarmos o conteúdo de uma variável, utilizamos o comando **echo** conforme mostrado anteriormente e o símbolo **$** na frente do nome da variável.
- Exemplo
  - var=”esta é uma mensagem”
  - echo “O conteúdo da variavel é $var”
  - echo O conteúdo da variavel é $var
- Para apagar a variável utilizamos o comando **unset**
- Exemplo
  - unset var
  - echo $var

Variáveis

- Podemos atribuir um valor nulo a uma variável da seguinte forma
  - var=”” ou
  - var=
  - echo $var
- Através do comando **set**, podemos verificar a existencia de todas as variáveis de ambiente. Como existem muitas variáveis de ambiente que serão abordadas adiante, utilizamos **PIPELINE** para realizar a paginação da saída do comando **set**.
- Exemplo
  - set | more ou set | pg

Variáveis

- Mesmo sem o usuário saber, diversas variáveis são criadas para auxiliar o funcionamento da Sessão.
- Exemplo de Variáveis que são criadas no inicio da sessão
  - BASH=/bin/bash
  - BASH_VERSINFO=([0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="i486-slackware-linux-gnu")
  - BASH_VERSION='2.05b.0(1)-release'
  - HOME=/root
  - HOSTNAME=pintagol.fespnet
  - HOSTTYPE=i486

Variáveis

- Existem quatro de váriaveis que são utilizadas em uma Sessão
  - Variáveis definidas por Usuário
    - São as variáveis **definidas pelo usuário durante** a Sessão. Exemplo: var, data
  - Variáveis do ambiente do Usuário
    - São as variáveis que **criadas no inicio da abertura** da Sessão do usuário, quando executado os arquivos profile e .profile . Exemplo HOME, USERID, etc.
  - Variáveis de estado de processos
    - São variáveis guardam status de processos executados na Sessão corrente. Exemplo: $?, $!, etc.
  - Variáveis acessíveis como argumentos
    - São variáveis que guardam os valores que são passados para um Shell Script na forma de parametros. Exemplo: $1, $#, etc.

Exercício 3

- Definir uma variável X com o valor 10 e mostrar na tela
- Definir uma varialve XN cujo conteúdo deve ser “Estou aprendendo Shell Script” e a seguir mostrar na tela
- Apagar as variáveis X e XN criadas anteriormente

Realizando Cálculos

- Através do prompt do Shell podemos realizar cálculos utilizando o comando **expr**
- Exemplo
  - expr 2 + 2 (soma)
  - expr 2 – 3 (subtração)
  - expr 2 * 3 (multiplicação)
  - expr 2 % 3 (resto de divisão)
  - expr 2 / 3 (inteiro da divisão)

A Crase

- O caractere crase (`) é utilizado quando queremos acessar a saída padrão (guardando um resultado) de um comando em Shell Script.
- Exemplo (1) - echo A soma de 2 + 2 é `expr 2 + 2`
  - O resultado será a string : “A soma de 2 = 2 é 4
- Exemplo (2) - var=`expr 2 + 2`
  - A variável **var** será iniciado com o valor 4
- Exemplo (3) - hoje=`date +%d/%m/%y`
  - A variável hoje será iniciado no formato ddmmaa
- Exemplo (4) – touch arq.h`date +%H%M%S`
  - Será criado o arquivo arq.hhhmmss corrente (arq.h120144)

Exercício 4

- Mostrar no monitor a soma de 4 e 15
- Mostrar no monitor a multiplicação de 4 e 15
- Mostrar no monitor a subtração de 4 e 15
- Mostrar no monitor o resto da divisão de 15 por 4
- Mostrar no monitor o inteiro de divisão de 15 por 4
- Atribuir a variável X o resto da divisão de 15 por 4
- Atribuir a variável XY o inteiro da divisão de 15 por 4
- Atribuir a variável V a quantidade de arquivos no diretório corrente (utilize os comandos ls, wc)

Variáveis de Usuário

- São inicializadas e referenciadas pelo usuário
- Somente o usuário e o Shell sabem de sua existencia
- Estão associadas ao Shell que a criou
- Não é automático a passagem para outro Shell
- As variáveis não são enviadas para o processo Shell filho e vice-versa.

Variáveis de Usuário

- Para verificar o funcionamento do Shell seguir os passos indicados:
  - Criar uma variável V=10
  - Mostrar o seu conteúdo – echo $V
  - Escrever o Script abaixo e executar,(ou executar um novo bash que tem o mesmo efeito)

```
#!/bin/bash
echo “O Valor de V é $V”
```

Variáveis de Usuário

- Como podemos verificar, **não** foi mostrado o contéudo da variável V, isso ocorre porque para execução do Script, um novo Shell é criado (processo filho) para execução do Script. A variável criada em um Shell não é vista em outro.
- Para que uma variável criado em um Shell seja acessível para todos **Shell Filho**, é necessário utilizar o comando **export**.
  - Exemplo: export V
- O comando **export** torna GLOBAL a variável
- Para confirmar o resultado, executar o procedimento anterior.

Variáveis de Ambiente

- Todas as variáveis de ambiente são definidas no início da Sessão do Usuário
- Variáveis de ambiente comuns a todos os usuários são encontrados no arquivo /etc/profile
- Variáveis de ambiente de usuário são encontrados no arquivo ~HOME/.profile
- Primeiro é lido o arquivo /etc/profile e depois o arquivo ~HOME/.profile
- Para acessar as variáveis do ambiente utilizar o comando **set**

Variáveis de Processo

- $_ - Contém o último comando executado.
- $? - Contém o Status retornado pelo último comando executado
- $$ - Número de Identificação (PID) do processo Shell que está sendo executado
- $! - Número de Identificação (PID) do último processo executado em background

Variáveis de Processo

- Exemplos
  - ls -la
  - echo $? (volta 0)
  - ls -la *.xxxx (um arquivo que não existe, erro)
  - echo $? (volta 1 – erro)
  - ls -la & (emite um pid, ex. 234)
  - echo $! (volta 234)
  - echo $$ (volta o pid o bash, ex. 245)
  - ps (mostra processo bash, ex. 245)

Variáveis de Processo

- A variável **$?** guarda o status de final de execução do último comando executado.
- Quando executado um Shell Script também podemos modificar esta variável através do comando **exit**.
- Exemplo

```
#!/bin/bash

exit 3
```

No prompt de comando após a execução no script acima.

```
$ echo $?
```

Variáveis com Argumentos

- São variáveis que são utilizadas para referenciar argumentos passados para um procedimento Shell
- Essas variáveis são inicializadas pelo Shell na execução do comando
- As variáveis são:
  - $# - Número de parametros posicionais escritos na linha de comando
  - $0 – Nome do Script que está sendo executado
  - $1 .... $9 – Parâmetros posicionais
  - $* - Lista que contém o conjunto dos argumentos passados para a rotina Shell.
-

Variáveis Argumento

```
#!/bin/bash
echo “Voce passou $# parametros”
echo “O parametro zero é o nome da rotina que e $0”
echo “O primeiro parametro é $1”
echo “O segundo  parametro é $2”
echo “A lista de parametros é $*”
```

Salvar o script com o nome exec0.sh
Executar com o seguintes parametros
bash exec0.sh um dois tres 4 5 6 7 8 9 0 1 2 3

Shift de Argumentos

- Como visto anteriormente, podemos acessar somente nove variáveis através das variáveis $1 e $9
- Se tivermos mais que nove argumentos teremos que utilizar o **shift**
- O comando **shift** desloca os argumentos posicionais de uma posição para a esquerda, ou seja $2 será associado para $1, $3 para $2 e assim sucessivamente.
- O comando **shift** decrementa o valor da variável **$#** e não afeta **$0** que contém sempre o nome do procedimento.

Shift de Argumentos

```
#!/bin/bash
echo "Voce passou $# parametros"
echo "O parametro zero é o nome da rotina que e $0"
echo "O primeiro parametro é $1"
echo "O segundo  parametro é $2"
echo "Estou executando o SHIFT" nas variáveis
shift
echo "Agora tenho $# parametros"
echo "O parametro zero é o nome da rotina que e $0"
echo "O primeiro parametro é $1"
echo "O segundo  parametro é $2”
```

Salvar o script com o nome exec1.sh
Executar com o seguintes parametros
bash exec1.sh um dois tres 4 5 6 7 8 9 0 1 2 3

Comando Read

- Para ler o teclado e armazernar o que foi digitado em variáveis, utilizamos o comando **read**.
- Exemplo: read nome_de_variavel
  - read var
  - read varx vary varz
- O comando **read** permite que uma sequencia de caracteres digitados sejam armazenadas em uma variável
- A cada espaço digitado, o comando **read** procura a próxima variável, até que elas se esgotem, ficando o restante da linha na última variável.

Comando read

```
#!/bin/bash
echo -n "digite uma variavel: "
read var1 var2 var3
echo "Primeira variavel --> $var1"
echo "Segunda variavel  --> $var2"
echo "Terceira variavel --> $var3”
```

Salvar o script com o nome exec2.sh
Executar o script

Controle de Fluxo

- Até o momento todos os comandos foram executamos sequencialmente.
- O Shell permite a utilização de comandos de controle de fluxo
  - Decisão
    - if
    - case
  - Repetição
    - while
    - for

Condicional

if [condição]

then

Se condição é verdadeira execute todos os comandos até fi

...

fi

** condição é a comparação entre dois valores

** observar sempre o formato [xcondiçãox], onde x corresponde a um espaço.

Condicional

```
if [ condition ]
then
        se condição for verdadeira
        execute todos os comandos até o else

else
        se condição não é verdadeira
        execute todos os comandos até o fi
fi
```

Condicional

```
if condition
        then
                if condition
                then
                        .....
                        do this
                else
                        ....
                        do this
                fi
        else
                 .....
                do this
        fi
```

Condicional

```
if condition
then
                se condição é verdadeira
                execute todos os comandos até  elif
elif condition1
then
                se condition1 é verdadeiro
                execute todos os comandos até  elif
elif condition2
then
                se condition2 é verdadeiro
                execute todos os comandos até  elif
else
                Nenhuma das condições acima são
verdadeiras
                execute todos os comandos até  fi
fi
```

Operadores de Comparação

Numéricos	
-eq	igual
-ne	diferente
-le	menor ou igual
-lt	menor que
-ge	maior ou igual
-gt	maior que

Alfanumérico	
=	igual
!=	diferente
-z	string sem conteúdo
-n	string com contéudo

Operadores de Comparação

Comparação matemática	Comparação em Script
5 == 6	if [5 -eq 6]
5 != 6	if [5 -ne 6]
5 < 6	if [5 -lt 6]
5 <= 6	if [5 -le 6]
5 > 6	if [5 -gt 6]
5 >= 6	if [5 -ge 6]

Operadores de Condição

Comparação entre String	Significado
if [string1 = string2]	string1 é igual a string2
if [string1 != string2]	string1 não é igual a string2
if [-z string1]	string1 é nulo e existe
if [-n string1]	string1 não é nul e existe

Teste com arquivos e diretórios

Teste	Significado
if [-s arquivo]	o arquivo não está vazio
if [-f arquivo]	é um arquivo normal e não é diretório
if [-d arquivo]	é um diretório
if [-w arquivo]	é um arquivo para gravação
if [-r arquivo]	é um arquivo read-only
if [-x arquivo]	é um arquivo executável

Operadores Lógicos

Operadores lógicos são utilizados para combinar duas ou mais condições

Operador Lógico	Significado
! expressão	operador lógico NOT
expressão1 **-a** expressão2	operador lógico AND
expressão1 **-o** expressão2	operador lógico OR

Controle de Fluxo - Exemplos

O exemplo abaixo ilustra a comparação de valores numéricos.

```
#!/bin/bash
echo -n "digite a variavel A: "
read var1
echo -n "digite a variavel B: "
read var2
if [ $var1 -eq $var2 ]
then
    echo "var1 é igual a var2"
else
    echo "var1 não é igual a var2"
fi
```

Controle de Fluxo - Exemplo

O exemplo abaixo ilustra a comparação de valores alfanuméricos.

```
#!/bin/bash
varx="Bom dia"
vary=dia

if [ "$varx" = "Bom dia" -a $vary = dia ]; then
   echo "A comparação de String é verdadeira"
else
   echo "A Comparação de Striong é falsa"
fi
```

Controle de Fluxo - Exemplos

```
#!/bin/bash

echo -n "digite o nome do arquivo: "
read var1

if [ -f "$var1" ]
then
        echo "$var1 é um arquivo normal"
        if     [ ! -s "$var1" ]; then
               echo "Este arquivo está vazio"
        else
               echo "Este arquivo não está vazio"
        fi
        if    [ -r "$var1" ]; then
              echo "O arquivo tem permissão de leitura"
        else
              echo "O arquivo não tem permissão de leitura"
        fi
else
         if       [ -d "$var1" ]; then
                  echo "$var1 é um diretório"
         fi
fi
```

Controle de Fluxo - Exemplo

```
#!/bin/bash

echo -n "digite o valor do número: "
read var1

if [ $var1 -eq 1 ]
then
    echo "Vc escolheu o Valor um"
elif [ $var1 -eq 2 ]; then
    echo "Vc escolheu o valor dois"
elif [ $var1 -eq 3 ]; then
    echo "Vc escolheu o valor três"
else
    echo "Vc escolheu uma opção inválida"
fi
```

Decisão com Múltiplas Escolha

```
case  $variável  in
          padrão1)   commando
                     ...
                     ..
                     commando;;
          padrão2)   command
                     ...
                     ..
                     commando;;
          padrãoN)   commando
                     ...
                     ..
                     commando;;
          *)         commando
                     ...
                     ..
                     commando;;
 esac
```

Decisão com Múltiplas Escolhas

```
#!/bin/bash

echo -e n"Menu de Escolha \n\n"
echo "1 - Inclusão "
echo "2 - Consulta"
echo "3 - Exclusão"
echo "4 - Alteração"
echo -e "5 - Fim\n"
echo -n "Escolha uma opção -> "
read op
case $op in
     1)echo "Vc escolheu a opção Inclusão";;
     2)echo "Vc escolheu a opção Consulta";;
     3)echo "Vc escolheu a opção Exclusão";;
     4)echo "Vc escolheu a opção Alteração";;
     5)echo "Vc escolheu a opção Fim";;
     *)echo "Vc escolheu uma opção inválida";;
esac
```

Decisão com Múltiplas Escolhas

```
#!/bin/bash

echo -e n"Menu de Escolha \n\n"
echo " Inclusão "
echo " Consulta"
echo " Exclusão"
echo " Alteração"
echo -e " Fim\n"
echo -n "Escolha uma opção -> "
read op
case $op in
      “Inclusao”)echo "Vc escolheu a opção Inclusão";;
      “Consulta”)echo "Vc escolheu a opção Consulta";;
      “Exclusao”)echo "Vc escolheu a opção Exclusão";;
      “Alteracao”)echo "Vc escolheu a opção Alteração";;
      “Fim”) echo "Vc escolheu a opção Fim";;
      *)echo "Vc escolheu uma opção inválida";;
esac
```

Repetição - for

```
for  variable in lista_de_valores
do
        Executa todos os comandos para cada elemento da
        lista.
Done
```

Exemplo

```
#!/bin/bash

for i in 1 2 3 4 5 6 7 8 9 0
do
    echo $i
done

lista="1 2 3 4 5 6 7 8 9 0"
for i in $lista
do
   echo $i
done
```

Repetição - for

```
for (( expr1; expr2; expr3 ))
do
        repete todos os comandos enquanto
        expr2  seja verdadeiro
done
```

Exemplo:

```
#!/bin/bash

for ((i = 0; i<= 5; i++ ))
do
   echo $i
done
```

Repetição - while

```
while [ condição ] ou [ true ] ou [ : ]
do
        Todos os comandos serão executados enquanto
         a condição é verdadeiro.

        command1
        command2
        command3
            ..
done
```

Exemplo:

```
#!/bin/bash

cont=0
while [ $cont -le 40 ]
do
   echo $cont
   cont=`expr $cont + 1`
done
```

Repetição

- Podemos interferir no processo de repetição (interromper ou continuar) utilizando os comandos continue e break.
- O comando **continue** força uma nova análise da condição (volta para o inicio da repetição)
- O comando **break** interrompe definitivamente o laço

Repetição - Continue

```
#!/bin/bash
numero=0
while [ $numero -lt 11 ]
do
    echo -n "Digite um numero de 0 a 10: "
    read numero
    if [ "$numero" -gt 10 ]
    then
        continue
    fi
    if [ "$numero" -lt 11 ]
    then
        echo "Voce digitou $numero"
    fi
done
echo “final de procedimento”
```

No exemplo acima, todo número digitado entre 0 e 10 será apresentado na tela e o laço será continuado. Quando for digitado um número maior que 10, no meio do procedimento, o laço é desviado para o início e a seguir é encerrado.

Repetição - break

```
#!/bin/bash
numero=0
while [ $numero -lt 11 ]
do
    echo -n "Digite um numero de 0 a 10: "
    read numero
    if [ "$numero" -gt 10 ]
    then
        break
    fi
    if [ "$numero" -lt 11 ]
    then
        echo "Voce digitou $numero"
    fi
done
echo “final de procedimento”
```

No exemplo acima, todo número digitado entre 0 e 10 será apresentado na tela e o laço será continuado. Quando for digitado um número maior que 10, no meio do procedimento, o laço é encerrado.

Execução condicional

- Os operadores && (and) e || (or) condicionam a execução de um comando com base na finalização de um comando anteriormente executado.
- Sintaxe
  - comando1 && comando2
  - comando2 será executado somente se comando1 retornar um status de saída igual a zero ($? = 0)
  - comando1 || comando2
  - Comando2 será executado somente se comando1 retornar um status de saída diferente de zeros ($? != 0)

Execução Condicional

- ls *.txt && echo “Achei os arquivos procurados”
- No exemplo acima, **se existir** algum arquivo com a extensão **txt**, será mostrado a mensagem, caso contrário nada será mostrado
- ls *.txt || echo “Não achei os arquivos procurados”
- No exemplo acima, **se não existir** arquivos com a extensão **txt**, será mostrado a mensagem, caso contrário nada será mostrado.
- ls *.txt && echo “Achei os arquivos procurados” || echo “Não achei os arquivos procurados”
- O exemplo acima une as duas contruções apresentadas anteriormente.

Funções

- Uma função é um conjunto de comandos que realizam uma tarefa única.
- O Shell permite a criação de funções que são chamadas durante a execução do script
- Sintaxe:

```
nome_de_função ()
{
    comando1
    comando2
    comando3
    ...
    return
}
```

Funções

```
#!/bin/bash
function mostra_valores()
{
  echo "Valor1: $var1"
  echo "Valor2: $var2"
  return
}

function mostra_soma()
{
  echo "$var1 + $var2 = `expr $var1 + $var2`"
  return
}

# Call the function
clear
echo -n "Valor 1: "
read var1
echo -n "valor 2: "
read var2
mostra_valores; mostra_soma
```

Funções – Passagem de Parametros

```
#!/bin/bash
function demo()
 {
        echo "Todos os argumentos da função demo(): $*"
        echo "Primeiro argumento -  $1"
        echo "Segundo argumento -  $2"
        echo "Terceiro argumento  -  $3"
        return
 }
 #
 # Chamada da função
 #
demo -f foo bar
```

Retorno de Valor

```
#!/bin/bash
function cal()
{
    n1=$1
    op=$2
    n2=$3
    ans=0
    if   [ $# -eq 3 ]; then
          ans=`expr $n1 $op $n2 ))
          echo "$n1 $op $n2 = $ans"
          return $ans
    else
          echo "Função cal necessita de três argumentos"
    fi
    return
 }
#chamada de função
 cal 5 + 10
 cal 10 - 2
 cal 10 / 2
 echo $?
```

Funções - Observação

- Variáveis em Shell Scripts são globais
- Passagem de parametros obedece as variáveis de ambiente $1 a $9
- O valor retornado de uma função é armazenado na variável de processo $?

RESPOSTAS

Exercício 1.
find / -amin 10 > hoje.txt &
find / -iname "*.cfg" > cfg.txt &
cat < hoje.txt > total.txt
cat < cfg.txt >> total.txt

Exercício 2.
ls -la | more
who | sort
who | wc -l
cat < /etc/passwd | sort
cat < /etc/passwd | sort > cadastro.txt
who | grep “raul”

Respostas

- Exercício 3
  - X=10
  - Echo $X
-
  - XN=“Estou aprendendo Shell Script”
  - Echo $XN
-
  - Unset X
  - Unset XN

Respostas

- Exercício 4.
- echo `expr 4 + 15`
- echo `expr 4 \* 15`
- echo `expr 4 - 15`
- echo `expr 15 % 4`
- echo `expr 15 / 4`
- X=`expr 15 % 4 `
- XY=`expr 15 / 4 `
- V=`ls -la | wc -l`