LivreCamp

Minicurso Shell Script

Shell Script - O que é?

- Shell Script é uma linguagem de script com diferentes dialetos dependendo do interpretador.
- Executando seu script:

```
$ ./nome_do_script.sh
```

- Necessário dar permissão de execução com a seguinte linha de comando:

```
$ chmod +x nome_do_script.sh
```

Shell Script - Por que aprender?

- Automatização de tarefas repetitivas
  - Instalação de softwares
  - Testes automatizados
  - Configuração de máquinas
- Usar todos as linhas de comando de maneira contínua, com repetições e tomadas de decisão (guarde bem essa informação)
  - if
  - for
  - while
- Manutenção da saúde do sistema
  - kill
  - subir aplicações
- Busca de arquivos e execução de tarefas sobre eles
- E, até mesmo, gerenciar suas séries, que vamos aprender hoje...

#!/bin/sh ?

Shebang - O que é ?

- A primeira linha do script sempre é algo como:
  - `#!/bin/sh`
  - `#!/bin/bash`
- Essa primeira linha diz qual interpretador vai ser usado para rodar os comando que estão dentro do seu script, sendo o Bourne Shell no primeiro e o Bash Shell, no segundo caso.

LivreSeries

Vamos começar?

Shell Script - Comandos básicos

- ls - listar conteúdo de um diretório
- cd - muda a localização do usuário
- grep - procura pro padrões em arquivos ou saída
- echo - printa na tela alguma saída
- read - lê entradas e atribui a variáveis
- | - pipe para direcionamento de saída
- if - condicional
- >>, > - saída em arquivo
- cut - corta saída por campos
- mkdir - cria diretórios

Mas pode ficar calmo que vamos ensinar como usar cada um de maneira automatizada

Exercício

Exercício - Criando o diretório do Projeto LivreSeries

- Abra o terminal e navegue para onde você deseja salvar o projeto, crie uma pasta chamada LivreSeries e entre nessa pasta. Dica: mkdir, cd.

Exercício - Criando o diretório e arquivo do Projeto LivreSeries

- Abra o terminal e navegue para onde você deseja salvar o projeto, crie uma pasta chamada LivreSeries e entre nessa pasta. Dica: mkdir, cd.

```
$ cd caminho/da/pasta
```

```
$ mkdir LivreSeries
```

```
$ cd LivreSeries
```

- Crie o um arquivo que será o seu gerenciador de séries. Faça isso da maneira que achar melhor. Se quiser, pode fazer isso pela linha de comando da seguinte maneira:

```
$ touch LivreSeries.sh
```

Exercício - Tornando o script executável

- Para dar permissão de execução ao script, digite:

```
$ chmod +x LivreSeries.sh
```

- Pronto, agora você tem permissão para executar o arquivo. Mas calma que ainda não codamos nada para executá-lo.

Comandos Básicos - read e echo

- **Read:** comando usado para pegar entrada do usuário e atribuir seu valor a uma variável, para poder reutilizar dentro do seu script.

```
read -p “Mensagem para o usuário “ variavel_que_recebera_entrada
```

- Para estabelecer um paralelo entre Shell Script e C, essa o **read** tem a mesma função que o **scanf** do C.
- **Echo:** função responsável por “printar” alguma mensagem, pode misturar mensagens e valores de variáveis.

```
echo “Meu nome eh $nome”
```

- Como mostrei acima, quando você deseja acessar (e não atribuir) um valor de variável, é necessário usar o $.

Exercício - Obtendo uma entrada e printando ela de maneira formatada

- Agora, usando o **echo** e o **read** que vimos anteriormente, implemente no seu programa **LivreSeries**, a leitura das variáveis do nome da série, temporada e episódio. Após isso, imprima da seguinte maneira:

```
game_of_thrones, temporada 7, episodio 5
```

- Não use espaços, para facilitar, use _.
- Não se esqueça de colocar #!/bin/sh na primeira linha, dar permissão de execução antes de executar.

Exercicio - Obtendo uma entrada e printando ela de maneira formatada

- Agora, usando o **echo** e o **read** que vimos anteriormente, implemente no seu programa **LivreSeries**, a leitura das variáveis do nome da série, temporada e episódio. Após isso, imprima da seguinte maneira:

```
game_of_thrones, temporada 7, episodio 5
```

- Não use espaços, para facilitar, use _.
- Não se esqueça de colocar #!/bin/sh na primeira linha, dar permissão de execução antes de executar.

```
read -p “Insira o nome da série” serie

echo “$serie, temporada $temp, episodio $epi”
```

Comandos Básicos - >>

- **>> :** responsável por direcionar a saída de um comando para um arquivo texto. Escreve a saída a partir da última linha do arquivo, não sobrescrevendo o mesmo.

```
comando >> arquivo_de_saida
```

- **> :** também possui a mesma função que o >> , porém sobrescreve a saída, apagando tudo o que estava escrito anteriormente.

```
comando > arquivo_de_saida
```

Exercício - Inserindo a entrada do usuário no seu txt

- Usando os redirecionadores de saída que acabamos de ver, faça a saída que fizemos no exemplo anterior (nome, temporada e episódio), para um arquivo chamado **series.txt.**
- Obs: o arquivo não precisa estar criado, o > ou >> irão criá-los automaticamente.

Exercício - Inserindo a entrada do usuário no seu txt

- Usando os redirecionadores de saída que acabamos de ver, faça a saída que fizemos no exemplo anterior (nome, temporada e episodio), para um arquivo chamado **series.txt.**
- Obs: o arquivo não precisa estar criado, o > ou >> irão criá-los automaticamente.

```
echo “$serie, temporada $temp, episodio $epi” >> series.txt
```

Comandos Básicos - If

- **If :** condicional do shell. Permite a tomada de decisões dentro do script.

```
if [ $variavel = “game_of_thrones” ]
then
     echo “muito bom gosto”
elif [ $variavel = “breaking_bad” ]
then
     echo “muito bom gosto também”
else
     echo “know nothing”
fi
```

Comandos Básicos - **If**

- Vale ressaltar que deve existir espaço entre as [ ] e as palavras que são escritas dentro delas.
- Além disso, existem opções de comparação e tomada de decisão muito poderosas em Shell Script, que podem ser acessadas por esse link:
    - http://codewiki.wikidot.com/shell-script:if-else

- Algumas dessas opções incluem verificar se um arquivo existe, se é um arquivo, se é executável, entre outras opções.

Exercício - Perguntando o que o usuário deseja fazer

- Usando o que acabamos de aprender, crie uma estrutura condicional no seu programa que pergunte ao usuário se ele deseja (I)nserir, (C)onsultar ou (E)xcluir uma série. Se ele desejar inserir, faça aquele **echo** que escrevemos anteriormente ir pra dentro do arquivo **series.txt**. Não se preocupe com os outros casos, pode só dar **echo** em alguma mensagem, faremos algo mais legal em breve.

Exercício - Perguntando o que o usuário deseja fazer

```
if [ $escolha = “I” ]
then
    echo “$serie, temporada $temp, episodio $epi” >> series.txt
elif [ $escolha = “C” ]
then
    echo “escolheu consulta”
elif [ $escolha = “E” ]
then
    echo “escolheu excluir”
fi
```

Comandos Básicos - **grep e cut**

- Para finalizar, vamos implementar as funções de retirada e consulta de uma série do **series.txt**
- Para isso vamos lançar mão de dois comandos:
    - grep
    - cut

- **grep:** procura por ocorrências de um padrão em uma saída ou arquivo.

```
grep padrão series.txt
```

- Possui a opção -v, que mostra tudo exceto o padrão.

Comandos Básicos - **grep e cut**

- **Cut:** como diz o nome, o **cut** corta a entrada a partir de um delimitador e permite que você acesse um campo específico.

```
cut -f 1 -d”,”
```

- O que estamos fazer acima é acessando o primeiro campo de algo delimitado por vírgulas. Para realizar essa operação em cima de uma variável e atribuir a uma outra, devemos fazer o seguinte processo:

```
resultado=$(echo $variavel | cut -f 1 -d”,”)
```

Exercicio Final - Implementando as funções faltantes

- Agora que você já aprendeu todas essas funções poderosas que o Shell Script pode oferecer, implemente as funções que estão faltando.
- Procure no **series.txt** a série que deseja e imprima de forma formatada seu nome, temporada e episódio.
- Exclua uma série usando as opções mostradas no grep.
- **Dica:** jogue a saída desse último comando em um arquivo auxiliar e depois renomeie este.
- A solução está em https://github.com/Guilhermeslucas/LivreSeries

Dúvidas?
Espero que tenham gostado.

Programando em Bash

#!/bin/mc102

Variáveis - Tipos

A princípio, tudo é string, a não ser que seja formada apenas por números.

> Strings (com ou sem aspas):

```
”palavra”, nome, a, 4784a, ...
```

> Números (apenas inteiros):

```
558, -8979, 0, “1”, ...
```

Variáveis - Atribuição

```
VARIAVEL=valor
```

Não pode ter espaços entre `=` e as palavras.

```
x=”-42”
```

```
nome=helio
```

Variáveis - Acesso

Para obter o valor, usamos o cifrão:

```
x=”-42”
```

```
echo  $x
```

```
-42
```

Sem cifrão, não funciona:

```
echo  x
```

```
x
```

Exercício

Dê seu nome a uma variável e imprima ela

Exercício

Dê seu nome a uma variável e imprima ela

```
nome=”tony”

echo $nome
```

Variáveis

Pegando o código retornado de um comando:

```
$(gcc codigo.c -o codigo)

echo $?

# imprime 0 se não teve erro

# senão, imprime código do erro
```

Exercício

Compare o retorno do gcc para quando ele compila os códigos:

```
int main(){
}
```

```
int main(){
```

Exercício

Compare o retorno do gcc para quando ele compila os códigos:

```
int main(){
}


> $(gcc a.c -o a)
> echo $?
0
```

```
int main(){



> $(gcc a.c -o a)
a.c:1:1: error: expected
declaration or statement at
end of input
 int main(){

> echo $?
1
```

if else

Comparação entre Strings:

```
if [ nome = ”unicamp” ]
    then echo “uhu”
else
    then echo “aff”
fi
```

if else

Comparação entre Números:

```
if [ $curso -eq 42 ]
    then echo "cc"
else
    echo "ec"
fi
```

`-eq` igual

`-ne` não igual

`-gt` maior

`-ge` maior igual

`-le` menor

`-lt` menor igual

if else

Comparação entre Números:

```
if (("$numero" == 42))
    then echo "cc"
else
    echo "ec"
fi
```

Exercício

Receba uma idade do input e diga se a pessoa é adulta

Exercício

Receba uma idade do input e diga se a pessoa é adulta

```
read idade

if (("$idade" >= 18))
    then echo "muito adulto você"
else
    echo "muito baby"
fi
```

for

O for tem várias maneiras de ser usado em listas, por exemplo:

```
for i in {1..10}
do
  echo "Loop ... número $i"
done
```

for

O for tem várias maneiras de ser usado em listas, por exemplo:

```
n=4

for (( i=1; i<=$n; i++ ))

do

   echo "Welcome $c times"

done
```

for

Os elementos da lista podem ser comandos:

```
for i in clear cd "ls -l"
do
  echo "comando $i:"
  $i
done
```

while

Basta colocar o teste entre [ ]:

```
INPUT_STRING=ola

while [ "$INPUT_STRING" != "tchau" ]

do

  echo "Diga algo: "

  read INPUT_STRING

  echo "Você disse: $INPUT_STRING"

done
```

case

Use * para o “default”:

```
var=4
case $var in
  2)
    echo "var é o número 2";;
  *)
    echo "var não é o número 2";;
esac
```

function

Todas as funções são void:

```
function print {
  echo "oi"
}
function repeat {
   echo $1
}
repeat “ola”
```

Exercício

Escreva uma função que renomeia um arquivo usando mv:

Exercício

Escreva uma função que renomeia um arquivo usando mv:

```
function rename {
    echo “Nome original:”
    read nome1
    echo “Nome novo:”
    read nome2
    mv $nome1 $nome2
}
```

Testador do Susy

Lab: https://github.com/andrealmeid/testador

1. Compilar o código
2. Rodar os testes
3. Apresentar o resultado ao usuário

Testador do Susy

1. Compilar o código

```
$(gcc lab.c -o lab)

if [ $? -ne 0 ]
    then
    echo "Erro na compilação"
    exit
fi

echo "Compilação OK"
echo ""
```

Testador do Susy

2. Rodar os testes
3. Exibir resultados

```
for file in testes/*.in; do
    echo Teste $file
    file="${file%.*}"
    ./lab < $file.in > $file.out
    $(diff $file.out $file.res > diff.txt)

    if [ $? -eq 0 ]
    then echo "OK"
    else
    echo "ERRO"
    cat diff.txt
    fi

    echo ""
done
```

Testador do Susy

Possíveis melhorias:

1. Outputs coloridos
2. Valgrind
3. Download dos testes

Sugestão de solução: https://github.com/jwnx/lab-ic

Dúvidas?