



Título do Curso:

Administração de Sistemas
GNU/Linux



Conteúdo do curso

ADMINISTRAÇÃO DE SISTEMAS GNU/LINUX

- ✓ Tópico 1: Introdução ao sistema operacional GNU/Linux.
- ✓ Tópico 2: Introdução ao Shell e comandos básicos.
- ✓ Tópico 3: Manipulação de conteúdos com comandos no Shell.
- ✓ Tópico 4: Comandos para gerenciamento do sistema e do Hardware.
- ✓ Tópico 5: Editor de Texto VI.
- ✓ Tópico 6: Administração de usuários e grupos.
- ✓ Tópico 7: Gerenciamento de permissões.
- ✓ Tópico 8: Gerenciamento de processos.
- ✓ Tópico 9: Sistemas de arquivos e particionamento.
- ✓ Tópico 10: Expressões regulares.
- **Tópico 11: Introdução ao Shell Script.**
- Tópico 12: Gerenciamento de Pacotes.
- Tópico 13: Agendamento de tarefas (cron) e Backup.





DGP

Tecnologia da Informação

Tópico 11

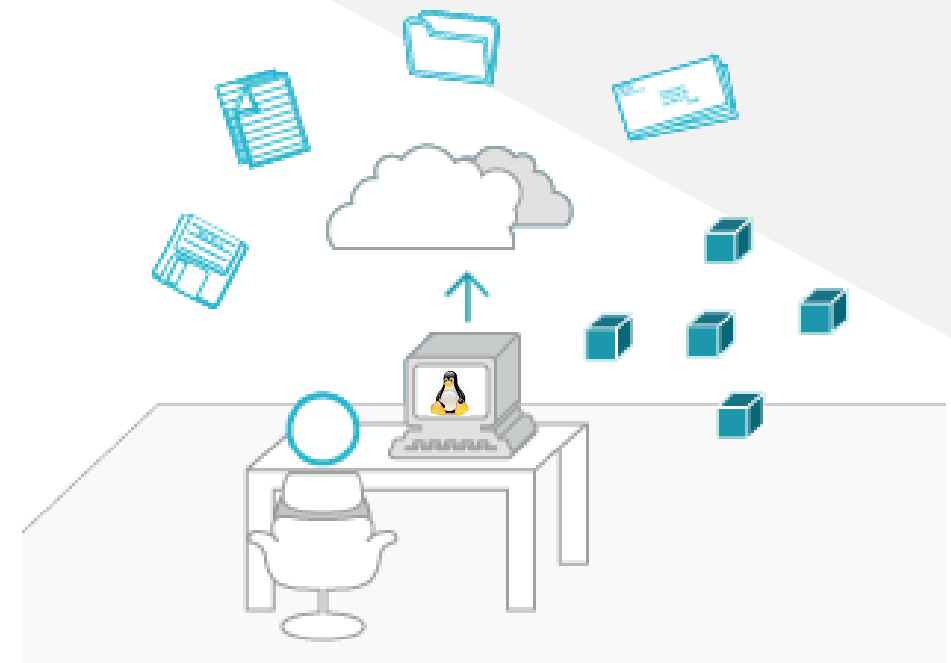
Introdução ao Shell Script



Introdução ao Shell Script

`#!/Shebang`

- Neste slide teremos conceitos e fundamentos sobre a estrutura e o funcionamento de um Shell Script:
 - Conceitos e aplicabilidade;
 - Criando um Shell Script;
 - Interação com o usuário através do Shell Script;
 - Definindo variáveis e programando em Shell Script.



Conceitos e aplicabilidade

Tópico 11: Introdução ao Shell Script.



Introdução a Shell Script

- Antes de iniciar o tema, vamos revisar alguns conceitos:
 - **Shell** → Interpretador de comandos Linux, ou seja, o “prompt” do Linux;
 - Atua entre o “**kernel**” do SO e o usuário, aguardando que novos comandos sejam executados pelo usuário;
 - **Script** → É um programa ou uma sequência de instruções que serão interpretadas ou realizadas por outro programa de computador:
 - Caso seja uma linguagem de script (**Perl**, **JavaScript**, **C**, **PHP**, entre outros), o código será interpretado por outro programa (geralmente um compilador);
 - Caso seja uma sequência de instruções ou comandos (como o **Shell** do Linux ou comandos do **Prompt** do M\$-DOS), o código é interpretado pelo próprio sistema ao invés de ser compilado.
- Logo, um **Shell Script** nada mais é do que um arquivo texto com diversos comandos em sequência, porém, com mais possibilidades...



Introdução a Shell Script

Algumas Possibilidades

- Na maioria dos casos, um **Shell Script** é utilizado para **automatizar tarefas**, **reduzindo** a necessidade de **intervenções humanas** para determinadas tarefas rotineiras;
- O **Shell Script** aceita **qualquer comando do Shell**;
- Porém, além de aceitar a execução de comandos existentes no Shell, podemos utilizar estruturas de **programação** em um Shell Script:
 - Estruturas de decisão → IF (famoso “se”) e o CASE;
 - Estruturas de repetição → FOR, WHILE e UNTIL;
 - Definição de variáveis → **\$VARIÁVEL**
- Além das estruturas de programação citadas acima, podemos criar menus e interfaces para prover uma interação com o usuário.



Introdução a Shell Script

#!/bin/bash

- Para possibilitar a identificação posterior do código (ou de parte dele, caso o Script seja extenso), é **extremamente recomendável** a inserção de “**comentários**” no código.
- Para inserir um **comentário**, basta adicionar o **#** no início da linha que o seu conteúdo não será lido/interpretado durante sua execução.
 - Exemplo:

```
#!/bin/sh
#
# This script will be executed *after* all the other init scripts.
# You can put your own initialization stuff in here if you don't
# want to do the full Sys V style init stuff.

touch /var/lock/subsys/local
loadkeys br-abnt2
```




Introdução a Shell Script

Shebang → **#!/bin/bash**

- Toda linha que tem o caractere **#** no início não é interpretada, exceto o “**Shebang**”, utilizado para informar qual **interpretador de comandos** deve ser utilizado para interpretar a sequência de instruções do Script;
 - **Shebang = #!** → Exemplos:
 - **#!/bin/bash** → O Shell Script será interpretado pelo **Shell Bash**;
 - **#!/bin/sh** → O Shell Script será interpretado pelo **Shell SH**;
 - **#!/usr/bin/awk -f** → O Script será interpretado pelo **comando AWK**;
- OBS.: A utilização do “**Shebang**” não é obrigatória, a menos que o script precise ser executado pelo próprio sistema como se fosse um comando binário. Caso o mesmo seja criado para um usuário executá-lo, ou adicionado em um script de inicialização do sistema (como o “**rc.local**”) que já possui o “**Shebang**”, não é necessário defini-lo.
 - Porém, veremos opções importantes oferecidas pelo “**Shebang**”;

Criando um Shell Script

Tópico 11: Introdução ao Shell Script.



Criando um Shell Script

- Para criar um **Shell Script**, devemos antes definir 4 itens:
 - Nome e local do arquivo/script;
 - Finalidade/Objetivo do Script;
 - Qual Shell deverá interpretar a sequência de comandos (Shebang);
 - Atribuir permissão de execução no arquivo;
- Portanto, vamos criar nosso primeiro Script:
 - Local: **/tmp/script_teste**
 - Finalidade: Informar a data atual e o usuário logado;
- Abaixo temos o código deste primeiro Script bem simples...

```
[root@server tmp]# cat script-teste
echo "A data atual é: `date`"
echo "O usuário logado é o: $USER"
```



Criando um Shell Script

- Observe os passos para a criação do Script, bem como sua execução:

```
[root@server ~]# cd /tmp/  
[root@server tmp]# vi script-teste  
[root@server tmp]# chmod +x script-teste
```

Script criado com o VI e permissão de execução atribuída com o chmod.

```
[root@server tmp]# ./script-teste  
A data atual é: Sun May 20 20:28:46 BRT 2012  
O usuário logado é o: root
```

Script executado e saída na tela.

- Primeiramente, devemos observar alguns detalhes:
 - O comando “echo” envia uma sequência de caracteres para a saída padrão;
 - A saída do comando “date” ficou um pouco “confusa” no meio da linha;
 - Foi possível realizar na mesma linha do “echo” a execução do comando “date” e a obtenção dos dados de uma variável do sistema, o “\$USER”;

```
echo "A data atual é: `date`"  
echo "O usuário logado é o: $USER"
```



Possíveis **problemas** na execução de um Shell Script

- “Caminho não encontrado”:

- A variável “\$PATH” define os locais dos arquivos executáveis no sistema (padrão), porém, o nosso script está em “/tmp”, portanto, devemos especificar o caminho completo, ou, apenas “./script” caso o diretório corrente (atual) seja o mesmo em que se encontra o script;
 - **Solução: Especifique o caminho correto/completo para executá-lo!**

```
[root@server tmp]# ./script-teste  
A data atual é: Sun May 20 20:28:46 BRT 2012  
O usuário logado é o: root
```

Executando no diretório corrente.

```
[root@server tmp]# /tmp/script-teste  
A data atual é: Sun May 20 20:29:02 BRT 2012  
O usuário logado é o: root
```

Especificando o caminho completo.



Possíveis **problemas** na execução de um Shell Script


- **“Permissão Negada”:**
 - Como o Script precisa ser **executado**, devemos atribuir permissões de **execução** no arquivo do Script;
 - **Atribua as permissões de execução (Ex.: `chmod +x script`)**
- **“Erro de Sintaxe”:**
 - O Script foi executado, porém, possui erros de sintaxe no arquivo de Script;
 - **Solução: Verifique a sintaxe dos comandos utilizados e possíveis “aspas” que foram abertas e não foram fechadas.**
 - OBS.: Um recurso importante neste caso é adicionar o parâmetro “**-x**” no “**Shebang**” (Caso seja “`/bin/bash`”). Este parâmetro faz com que cada comando executado no Script seja exibido na tela.
 - OBS.2: Também temos o parâmetro “**-e**” que interrompe a execução do Script caso ocorra algum erro (**`Return status != 0`**).



Melhorando a saída do Script


- Apesar do nosso primeiro script possuir apenas duas linhas, a saída na tela fica um pouco “confusa”, dificultando o entendimento.
 - *Imagine um Script com vinte... Trinta linhas...*
- Sabemos que o comando “echo” pode ser utilizado para enviar mensagens na saída padrão (tela), porém, o “echo” também pode ser utilizado para enviar **NADA**, ou seja, uma linha vazia.
- Observe que uma pequena mudança pode tornar o script mais legível:

Alterando o conteúdo do Script (linha com “echo” vazio)...



```
[root@server tmp]# cat script-teste
echo "A data atual é:"
date
echo
echo "O usuário logado é o:"
echo $USER
```

...para adicionar linhas em branco na saída.



```
[root@server tmp]# ./script-teste
A data atual é:
Sun May 20 20:56:27 BRT 2012

O usuário logado é o:
root
```

Interação com o usuário através do Shell Script

Tópico 11: Introdução ao Shell Script.



Interação com o usuário

- Através do **Shell Script**, podemos interagir com o usuário, solicitando confirmações, inserção de dados, entre outras possibilidades.

```
[root@server tmp]# cat status-S0
#!/bin/bash
echo "Este Script exibe informações sobre o sistema. Deseja continuar? [sn]"
read RESPOSTA
test "$RESPOSTA" != s && exit
echo "A data atual é: "
date
echo
echo "Informações sobre o sistema: "
uptime
```

- O comando **“read”** recebe os dados informados pelo usuário e o armazena em uma variável definida na mesma linha do comando (no caso acima, variável de nome **“RESPOSTA”**);
- O comando **“test”** verificou se o conteúdo da variável **“\$RESPOSTA”** era diferente **“!=”** de **“s”**, para continuar ou encerrar a execução do script.



Interação com o usuário

- Observe que ao digitar uma letra diferente de “s” o script é encerrado:

```
[root@server tmp]# ./status-S0
Este Script exibe informações sobre o sistema. Deseja continuar? [sn]
n
[root@server tmp]#
```

```
[root@server tmp]# ./status-S0
Este Script exibe informações sobre o sistema. Deseja continuar? [sn]
s
A data atual é:
Sun May 20 21:08:27 BRT 2012

Informações sobre o sistema:
21:08:28 up 4:06, 1 user, load average: 0.11, 0.09, 0.09
```

- Temos diversas possibilidades com o **Shell Script** e o uso do comando “test” para verificações. Veremos algumas possibilidades a seguir.



Documentando o Script

- Este script é bem simples, porém, em scripts maiores e mais complexos, “**documentar**” o seu conteúdo é **muito importante**, para possibilitar a análise do seu conteúdo em futuras manutenções ou modificações;
- É recomendável deixar espaços (linhas em branco) entre partes do código, bem como redigir **comentários concisos e esclarecedores**, tornando a **visualização** do código mais agradável e **inteligível**, além de **facilitar** a sua **manutenção**;
- Para inserir um **comentário**, basta inserir um “**#**” e o comentário em seguida, conforme citado no início deste slide.



Documentando o Script

- Segue um exemplo com o script elaborado anteriormente:

```
#!/bin/bash

# status-SO --> Exibe na tela informações sobre o sistema.
# Autor: Guilherme Rodrigues Pereira

# Solicita a confirmação do usuário e armazena na
# variável $RESPOSTA os dados informados pelo usuário.
echo "Este Script exibe informações sobre o sistema. Deseja continuar? [sn]"
read RESPOSTA

# Verifica se o usuário digitou "s" para continuar
test "$RESPOSTA" != s && exit

# O comando "date" exibe a data e hora do sistema
echo "A data atual é: "
date
echo

# O comando "uptime" exibe por quanto tempo o sistema
# está ligado, quantidade de usuários logados, e o "Load Average"
echo "Informações sobre o sistema: "
uptime
```

Definindo variáveis e programando em Shell Script

Tópico 11: Introdução ao Shell Script.



Definindo variáveis

- Através das **variáveis**, podemos armazenar os dados obtidos durante a execução de um Script. Estes dados podem ser obtidos através da interação do usuário ou da saída de algum comando;
- Para trabalhar com variáveis, temos basicamente 4 comandos:

- Sinal de “=” para **definir** a variável:

```
[root@localhost ~]# VARIAVEL="Variavel Teste"  
[root@localhost ~]#
```

- Comando “**echo**” para **exibir** o conteúdo da variável:

```
[root@localhost ~]# echo $VARIAVEL  
Variavel Teste
```

- Comando “**unset**” para **apagar** o valor de uma variável:

```
[root@localhost ~]# unset VARIAVEL  
[root@localhost ~]#  
[root@localhost ~]# echo $VARIAVEL  
[root@localhost ~]# _
```

- Comando “**env**” que exibe todas as variáveis do sistema.



Definindo variáveis em um Shell Script

- Definindo a **variável** através da interação do usuário, comando “**read**”:

```
# Solicita a confirmação do usuário e armazena na
# variável $RESPOSTA os dados informados pelo usuário.
echo "Este Script exibe informações sobre o sistema. Deseja continuar? [sn]"
read RESPOSTA
```

- Definir manualmente (geralmente no início do Script):
 - OBS.: Não deve haver “espaços” antes ou após o sinal de “=”.
 - OBS.2: Para obter os dados da saída de um comando utilizamos o subshell (crase ou “\$()”);

```
# Definindo variáveis
VAR1="Teste VAR1, com espaços e aspas"
VAR2=SemEspaço_SemÂspas
DATA=$(date)
echo $VAR1
echo $VAR2
echo "A data atual é: $DATA"
```

```
Teste VAR1, com espaços e aspas
SemEspaço_SemÂspas
A data atual é: Seg Mai 21 19:38:49 BRT 2012
```



O comando “test”

- Segundo JARGAS, o comando “test” é o “canivete suíço” do Shell, pois permite realizar diversos testes em números, textos/string e arquivos.
- Segue abaixo uma tabela resumida dos principais parâmetros:

	Testes em variáveis		Testes em arquivos
-lt	Núm. é menor que (LessThan)	-d	É um diretório
-gt	Núm. é maior que (GreaterThan)	-f	É um arquivo normal
-le	Núm. é menor igual (LessEqual)	-r	O arquivo tem permissão de leitura
-ge	Núm. é maior igual (GreaterEqual)	-s	O tamanho do arquivo é maior que zero
-eq	Núm. é igual (Equal)	-w	O arquivo tem permissão de escrita
-ne	Núm. é diferente (NotEqual)	-nt	O arquivo é mais recente (NewerThan)
=	String é igual	-ot	O arquivo é mais antigo (OlderThan)
!=	String é diferente	-ef	O arquivo é o mesmo (EqualFile)
-n	String é não nula	-a	E lógico (AND)
-z	String é nula	-o	OU lógico (OR)



O comando “test” e o Shell Script

- Podemos utilizar o comando “test” de duas formas:
 - Sintaxe 1 → Utilizar o comando “test” com os demais parâmetros desejados para realizar a verificação:

```
if test "$QTDE" -gt 15  
then
```

- Sintaxe 2 → Especificar a sintaxe entre colchetes sem escrever de forma explícita o comando “test” (tendo em vista que o que estiver entre colchetes, será verificado pelo comando “test”);

```
if [ "$QTDE" -gt 15 ]  
then
```

- OBS.: Em ambos os casos citados acima temos o mesmo resultado.

Programando um Shell Script básico com o “IF”

Tópico 11: Introdução ao
Shell Script.



Programando com o “IF”

- A instrução “if” usa uma condição para tomar a decisão.
- Geralmente, a condição testada será uma expressão booleana (True/False).
 - A sintaxe da instrução “if” possui a seguinte estrutura: se a condição for verdadeira [True], execute a(s) instrução(ões) a seguir.

```
if condição:  
    instrução(ões) indentada(s)
```

- Frequentemente, espera-se que o programa também faça algo quando a condição não for verdadeira [False]. Para este caso, temos a instrução “else”:

```
if condição:  
    instrução(ões) indentada(s)  
else  
    outras instrução(ões) indentada(s)
```



Programando com o “IF”

Scripts básicos para exemplificar algumas possibilidades...

- Contabiliza a **qtde** de objetos em um diretório:

```
#!/bin/bash
# Verifica a quantidade de arquivos e subdiretórios.
# Autor: Guilherme Rodrigues Pereira

# Solicita ao usuário qual diretório deve ser verificado.
# A opção "-e" em conjunto com o "\c" eliminam a quebra de linha.
echo -e "Informe o diretório do qual deseja saber a qtde de arquivos:" "\c"
read DIR

# Lista o diretório informado e contabiliza a QTDE de objetos.
# O resultado é armazenado na nova variável QTDE.
QTDE=$(ls -la $DIR | egrep -v "^\.{1,2}$" | wc -l)

# Informa ao usuário a QTDE de objetos dentro do diretório e se
# o diretório possui mais de 15 objetos ou não.
if test "$QTDE" -gt 15
then
    echo "O diretório $DIR possui mais de 15 objetos."
    echo "Total de $QTDE objetos."
else
    echo "O diretório $DIR possui menos de 15 objetos."
    echo "Total de $QTDE objetos."
fi
```



Programando com o “IF”

Scripts básicos para exemplificar algumas possibilidades...

- **Editor VIM.** Mesmo script do slide anterior, porém, o **VIM** atribui cores distintas de acordo com a finalidade de cada parte do código:

```
#!/bin/bash
# Verifica a quantidade de arquivos e subdiretórios.
# Autor: Guilherme Rodrigues Pereira

# Solicita ao usuário qual diretório deve ser verificado.
# A opção "-e" em conjunto com o "\c" eliminam a quebra de linha.
echo -e "Informe o diretório do qual deseja saber a qtde de arquivos:" "\c"
read DIR

# Lista o diretório informado e contabiliza a QTDE de objetos.
# O resultado é armazenado na nova variável QTDE.
QTDE=$(ls -la $DIR | egrep -v "^\.{1,2}$" | wc -l)

# Informa ao usuário a QTDE de objetos dentro do diretório e se
# o diretório possui mais de 15 objetos ou não.
if test "$QTDE" -gt 15
then
    echo "O diretório $DIR possui mais de 15 objetos."
    echo "Total de $QTDE objetos."
else
    echo "O diretório $DIR possui menos de 15 objetos."
    echo "Total de $QTDE objetos."
fi
```



Programando com o “IF”

Scripts básicos para exemplificar algumas possibilidades...

- Testa arquivos 1 → Apenas verifica se o caminho digitado pelo usuário é um arquivo comum ou não (caso seja um dispositivo, ou diretório):

```
#!/bin/bash
# Script que testa arquivos
# Autor - Guilherme Rodrigues

echo "Digite o nome do arquivo"
echo "a ser verificado:"
read CAMINHO

# Verifica se é um arquivo ou não
if test -f "$CAMINHO"
then
    echo "O $CAMINHO é um arquivo"
else
    echo "O $CAMINHO não é um arquivo"
fi

"testa-arquivos" 15L, 284C
```



Programando com o “IF”

Scripts básicos para exemplificar algumas possibilidades...

- Testa arquivos 2 → Identifica se é um arquivo, diretório, ou outro tipo de objeto (dispositivo por exemplo).

```
#!/bin/bash
# Script que testa arquivos
# Autor - Guilherme Rodrigues

echo "Digite o nome do arquivo"
echo "a ser verificado:"
read CAMINHO

# Verifica se é um arquivo ou não
if test -f "$CAMINHO"
then
    echo "O $CAMINHO é um arquivo"
elif test -d "$CAMINHO"
then
    echo "O $CAMINHO é um diretório"
else
    echo "O $CAMINHO não é um arquivo comum ou diretório"
fi

"testa-arquivos2" 19L, 371C
```



Programando com o “IF”

Scripts básicos para exemplificar algumas possibilidades...

- Testa arquivos 3 → Verifica se o caminho é válido antes de testar...

```
#!/bin/bash
# Script que testa arquivos
# Autor - Guilherme Rodrigues

echo "Digite o nome do arquivo"
echo "a ser verificado:"
read CAMINHO

# Verifica se é um arquivo ou não
if test -e "$CAMINHO"
then
    echo "O $CAMINHO existe"
    if test -f "$CAMINHO"
    then
        echo "O $CAMINHO é um arquivo"
    elif test -d "$CAMINHO"
    then
        echo "O $CAMINHO é um diretório"
    else
        echo "O $CAMINHO não é um arquivo comum ou diretório"
    fi
else
    echo "O $CAMINHO não existe"
fi

"testa-arquivos3" 24L, 471C
```




Outras opções...

- Além da estrutura básica de programação como o “**IF**”, temos a possibilidade de criar “funções” ou utilizar outras opções como o “**FOR**”, “**WHILE**”, “**CASE**” e “**UNTIL**”;
- Também temos a possibilidade de criar telas através do “**xdialog**”, possibilitando uma interação no modo gráfico, porém, o pacote não é nativo em grande parte das distribuições;
- Não vamos aprofundar nestes tópicos, o objetivo é compreender os conceitos e a aplicabilidade do Shell Script (mais informações podem ser obtidas nos links/referências).



No próximo slide...

- Tópico 12: Gerenciamento de Pacotes:
 - Conceitos e aplicabilidade;
 - Instalação de um pacote através do código fonte;
 - Instalação através de pacotes pré-compilados (".rpm", ".deb"...);
 - Instalação através de Gerenciadores de pacotes.



Referências

- Aurélio Marinho Jargas – Canivete Suíço do Shell (Bash) – 2011:
 - <http://aurelio.net/shell/canivete/>
- Carlos E. Morimoto – Programando em Shell Script – 2010:
 - <http://www.hardware.com.br/guias/programando-shell-script/>
- Hugo Cisneiros – Programando em shell-script – 2004:
 - http://www.devin.com.br/shell_script/
- Pablo Carlos de S. Furtado – Shell Script: Primeiros Conceitos – 2005:
 - <http://www.vivaolinux.com.br/artigo/Shell-Script-Primeiros-conceitos?pagina=1>



Referências

- BONAN, Adilson Rodrigues. **LINUX – Fundamentos, Prática & Certificação LPI**. Editora: Alta Books. RJ. 2010;
- PEREIRA, Guilherme Rodrigues. **Slides para aula expositiva**. Centro Universitário UNA.
- SILVA, Gleydson Mazioli. **Guia Foca GNU/Linux**. Disponível em: <https://guiafoca.org/>



DGP

Tecnologia da Informação

Obrigado!



Guilherme Rodrigues