Minicurso de Shell Script

Paulo Henrique Cuchi & Renan Samuel da Silva

COLMEIA
Grupo de Extensão em Software e Hardware Livre

Universidade do Estado de Santa Catarina

25 de Abril de 2015

O que é Shell?

O que é Shell?

O que é um script?

O que é Shell?

O que é um script?

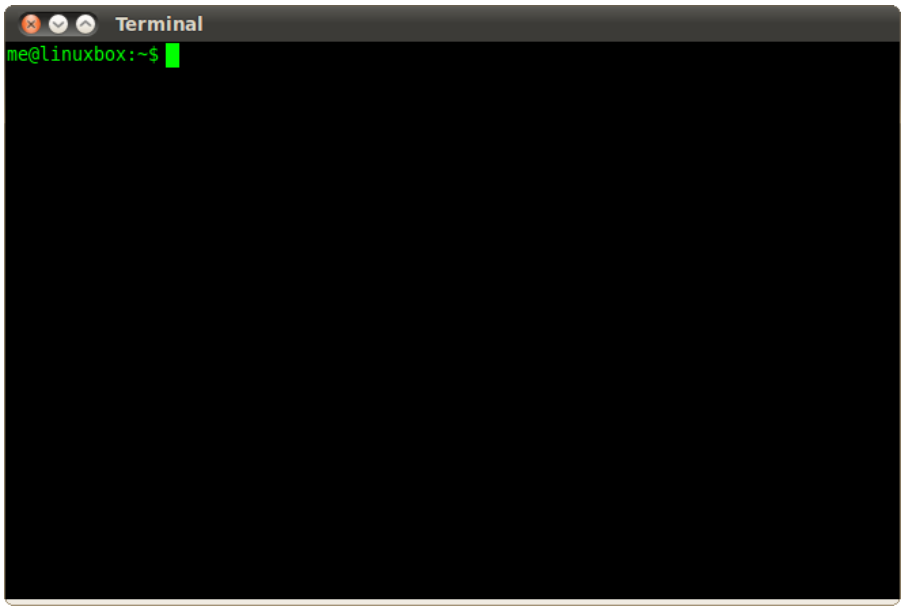
E **Shell Script**, o que é?

O Shell

Shell é o software que interpreta comandos, presente em sistemas operacionais da família *Unix*.

O Shell

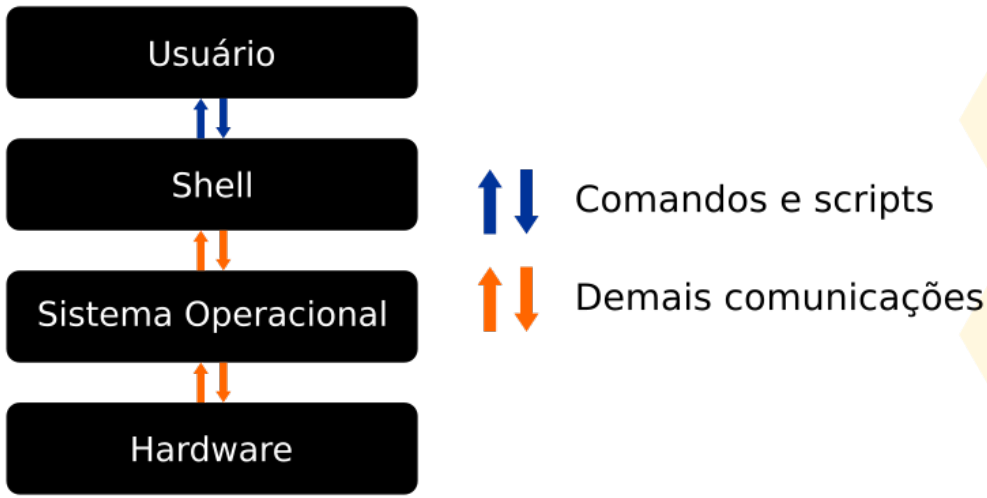
Shell é o software que interpreta comandos, presente em sistemas operacionais da família *Unix*.



Também é usualmente (e erroneamente, em alguns casos) chamado de *terminal*, *prompt de comando* ou simplesmente de *tela preta*.

O Shell

O shell funciona como uma das "pontes" entre o userspace e o sistema operacional.



Utilizando apenas comandos, é possível realizar qualquer rotina à nível de sistema.

O Shell

Como já mencionado anteriormente, o Shell é um componente que integra o sistema operacional, esse componente pode ter várias implementações diferentes.

Assim como distribuições, cada implementação é direcionada à projetos e tipos específicos de usuários.

O Shell

Algumas das implementações mais conhecidas são:

O Shell

Algumas das implementações mais conhecidas são:

- sh – Bourne Shell

O Shell

Algumas das implementações mais conhecidas são:

- sh – Bourne Shell
- bash – Bourne-again Shell

O Shell

Algumas das implementações mais conhecidas são:

- sh – Bourne Shell
- bash – Bourne-again Shell
- csh – C Shell

O Shell

Algumas das implementações mais conhecidas são:

- sh – Bourne Shell
- bash – Bourne-again Shell
- csh – C Shell
- ksh – KornShell

O Shell

Algumas das implementações mais conhecidas são:

- sh – Bourne Shell
- bash – Bourne-again Shell
- csh – C Shell
- ksh – KornShell
- zsh – Z Shell

O Shell

Algumas das implementações mais conhecidas são:

- sh – Bourne Shell
- bash – Bourne-again Shell
- csh – C Shell
- ksh – KornShell
- zsh – Z Shell
- fish – Friendly Interactive Shell

O Shell

Algumas das implementações mais conhecidas são:

- sh – Bourne Shell
- bash – Bourne-again Shell
- csh – C Shell
- ksh – KornShell
- zsh – Z Shell
- fish – Friendly Interactive Shell
- ...

Shell Script

Os interpretadores de shell também podem interpretar **arquivos**.

Esses arquivos são conhecidos como **scripts**.

Shell Script

Os interpretadores de shell também podem interpretar **arquivos**.

Esses arquivos são conhecidos como **scripts**.

Isso acaba transformando o shell em uma linguagem de script extremamente poderosa, assemelhando-se com *PHP*, *Python*, e *Lua*.

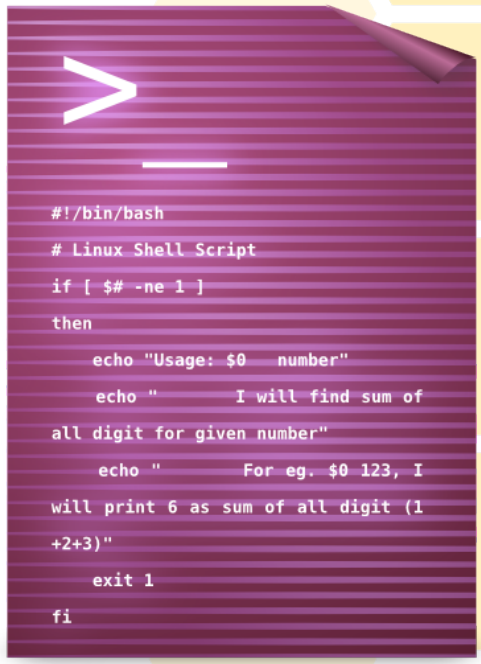
Shell Script

Os interpretadores de shell também podem interpretar **arquivos**.

Esses arquivos são conhecidos como **scripts**.

Isso acaba transformando o shell em uma linguagem de script extremamente poderosa, assemelhando-se com *PHP*, *Python*, e *Lua*.

Essa linguagem é o que chamamos de **shell script**.



Scripts em shell, **por quê?**

Scripts em shell, **por quê?**

- Comandos nativos para interagir com arquivos e o sistema

Scripts em shell, **por quê?**

- Comandos nativos para interagir com arquivos e o sistema
- Estruturas de linguagens imperativas, como **for**, **while** e **if**

Scripts em shell, **por quê?**

- Comandos nativos para interagir com arquivos e o sistema
- Estruturas de linguagens imperativas, como **for**, **while** e **if**
- ...e pra quem gosta, **Expressões regulares!**

Shell Script

Com shell script, é possível:

- Modificar uma grande quantidade de arquivos de uma só vez:
  - renomear
  - redimensionar (imagens)
  - gerar documentos de texto
- Automatizar rotinas no sistema
  - backups
  - limpeza de arquivos
  - atualizações
- E muito mais! :D

ls - list directory contents

Lista o conteúdo do diretório

- ls
- ls -a
- ls -la
- ls /home/um/diretorio/qualquer -la

pwd - print path of the working directory

Mostra o caminho do diretório atual

cd - change directory

Altera o diretório atual

cp - copy

Copia um arquivo ou diretório

- cp arquivo backups/
- cp musicas/ backups/ -rv
- cp certificado.pdf copia_certificado.pdf

mv - move/rename files or directories

Move ou renomeia um arquivo/diretório

rm - remove files or directories

Remove arquivos e diretórios

cat - concatenate files and print the output

Mostra o conteúdo de arquivos de texto

echo - display a line of text

Imprime uma linha de texto

Construindo Scripts

- Os Scripts em Shell são feitos em arquivos de texto, por convenção, usa-se a extensão **.sh**.
- É uma linguagem imperativa, possui a sintaxe semelhante com *perl* e *php*

Construindo Scripts

Exemplo de um script de shell:

```
#!/bin/bash
# <- Comentários começam com cerquilha
echo "Fazendo backup..."
cp musicas backup/ -r
cp videos backup/ -r
cp documentos backup/ -r
echo "Backup feito!"
```

Construindo Scripts

Assim como em grande parte das linguagens de script, o código é executado pela chamada do interpretador, nesse minicurso usaremos o **bash** por padrão:

```
bash meu_script.sh
```

O método acima executa qualquer script em Shell, mesmo que este não seja considerado um arquivo executável pelo sistema.

Construindo Scripts

Para executar um script em Shell da mesma maneira que se costuma executar um binário, é preciso definir o interpretador (shell padrão) na primeira linha do script:

```
#!/bin/bash
```

Essa linha é chamada de **hashbang**.
Em sistemas GNU/Linux e similares, ela é usada para definir o interpretador de qualquer linguagem de script.

Construindo Scripts

Mas isso ainda não é o suficiente! É preciso também "dizer" ao sistema que o script é um arquivo executável:

```
chmod +x meu_script.sh
```

Agora sim! podemos executar o script (da mesma maneira que se executa um programa):

```
./meu_script.sh
```

Variáveis

O Shell Script é uma linguagem de tipagem dinâmica, ou seja, não há a necessidade de declarar o tipo do identificador (variável).

Variáveis

O Shell Script é uma linguagem de tipagem dinâmica, ou seja, não há a necessidade de declarar o tipo do identificador (variável).

Exemplo

```
#!/bin/bash

curso="Minicurso de Shell Script"
meterial="~/udesc/Colmeia/minicurso_shell/"
duracao="3h30min"
n_participantes=30
```

Não pode haver espaços antes e depois do *igual*.

Variáveis

A chamada das variáveis é feita utilizando um cifrão (**$**) como prefixo.

Exemplo

```
#!/bin/bash

backups="/media/pendrive/backups/"

cp musicas/ $backups -r
cp videos/ $backups -r
cp documentos $backups -r
```

Entrada e Saída (I/O)

A entrada e saída dos dados é feita com os comandos **read** e **echo**.

Exemplo

```
#!/bin/bash

echo "Digite seu nome:"
read nome
echo "Bem-vindo ao minicurso de shell script, $nome!"
```

Entrada e Saída (I/O)

Para imprimir texto de maneira formatada, há também o comando **printf**.

Exemplo

```
#!/bin/bash

who="the hobbits"
where="Isengard"
printf "They are taking %s to %s!\n" $who $where
```

Programadores de **C** irão se sentir em casa :)

Variáveis

Variáveis podem ser atribuídas com saídas de comandos (que são strings), nesse caso, usamos os símbolos **$()**.

Exemplo

```
#!/bin/bash

agora=$(date)
echo $agora
```

Variáveis de Ambiente

No sistema, há as variáveis globais.

- Podem ser lidas por qualquer script
- Possuem informações úteis sobre o sistema e a sessão atual.

Essas variáveis podem ser vistas usando o comando **env**.

Variáveis de Ambiente

Exemplo

```
#!/bin/bash

echo "Nome do usuário: $USER"
echo "Pasta pessoal: $HOME"
echo "Idioma: $LANG"
```

Operações Aritiméticas

Em Shell Script, operações são feitas à partir do comando **expr**.

Exemplo

```
#!/bin/bash

echo "Digite dois valores:"
read x y
prod=$(expr $x \* $y)
echo "Produto de $x e $y: $prod."
```

O comando **expr** pode também ser usado para operações lógicas, *ver o manual*.

Operações Aritiméticas

Há também a expressão **$(( ))**, que faz a mesma coisa.

Exemplo

```
#!/bin/bash

echo "Digite dois valores:"
read x y
prod=$((x * y))
echo "Produto de $x e $y: $prod."
```

Por padrão, o **bash** só realiza operações com números inteiros.
Alguém sabe responder por quê?

Comando **test**

O comando test é bastante usado em estruturas condicionais e de repetição para testar a sanidade do sistema.

- Tal arquivo existe?
- É executável?
- É um diretório?

O comando **test** verifica tudo isso e muito mais! Ver o manual de referência:

```
man test
```

Estrutura Condicional - **if**

Diferente da maioria das linguagens, o shell script considera o **0** como verdadeiro e qualquer valor diferente como falso.

Isso vale também para as estruturas condicionais e de repetição.

Exemplo

```
#!/bin/bash

arquivo="/home/udesc/relatorio.txt"

# Se o arquivo existe...
if test -e $arquivo; do
    rm $arquivo
done
```

Estrutura de Repetição - **while**

O **while** repete o comando até ser falso (óbvio!).

exemplo

```
#!/bin/bash

# Script para conectar na maravilhosa rede da UDESC
while not netctl start alunos; do
    echo "Que droga! o sinal está ruim :/"
done
echo "Conectou! :D"
```

Estrutura de Repetição - **for**

O **for** itera uma lista.

Em Shell Script, uma lista não é nada mais que uma string separada por espaços, quebras de linhas ou *tabs*.

Exemplo

```
#!/bin/bash

hobbits="Frodo Sam Merry Pippin"
for h in $hobbits; do
    echo $h
done
```

Controle de fluxo

No *shell* é possível controlar o fluxo entre arquivos e programas.

>	Redireciona saída padrão para um arquivo
2 >	Redireciona saída de erros para um arquivo
& >	Redireciona saída padrão e de erros para um arquivo
>>	Redireciona a saída para o fim de um arquivo
<	Redireciona um arquivo para a entrada de um programa
|	Redireciona a saída de um programa para a entrada de outro

Visualização de arquivos

Para visualizar um arquivo use o comando **cat**:

- cat names.txt

O comando **head** e **tail** mostra o começo e fim de um arquivo:

- tail names.txt
- cat names.txt | tail
- cat names.txt | tail -n 25
- dmesg | head

Utilize **less** para navegar pelo arquivo:

- less names.txt
- dmesg | less

Contagem

wc é uma ferramenta capaz de contar:

-c	Caracteres
-w	Palavras
-l	Linhas
-L	Tamanho da maior linha

Exemplo:

- wc -c names.txt
- wc -L names.txt
- dmesg | wc -l

Ordenação

É possível ordenar um arquivo usando o comando *sort*:

sort	Ordena considerando os caracteres *ASCII*
sort -n	Ordena em ordem numérica
sort -d	Ordena considerando apenas letras e espaços
sort -r	Mostra na ordem inversa

Exemplo:

- sort ip2.txt
- sort ip2.txt -n
- sort ip2.txt -nr
- sort ip2.txt -r

Manipulação de strings

É possível cortar strings usando *cut*:

cut -c 5	Mostra o quinto carácter
cut -c 5-	Mostra do quinto em diante
cut -c -5	Mostra até o quinto carácter
cut -c 5-10	Mostra do 5 até o 10 carácter

Exemplo:

- echo 'GNU/Linux' | cut -c 4
- echo 'GNU/Linux' | cut -c -3
- echo 'GNU/Linux' | cut -c 5-
- echo 'GNU/Linux' | cut -c 6-8

Substituição de caracteres

O comando *tr* substitui ou remove caracteres arbitrários:

tr *set1 set2*	Substitui o *set1* pelo *set2*
tr -d set1	Remove o conteúdo do *set1*

[:digit:]	Todos os dígitos
[:alnum:]	Todos os números e letras
[:alpha:]	Todas as letras
[:lower:]	Todas as letras minusculas
[:upper:]	Todas as letras minusculas
[:punct:]	Sinais de pontuação

Exemplo:

```
echo 'LiNuX' | tr [:upper:] [:lower:]
```

O que é

- É um método formal de se especificar uma padrão de texto.
- Uteis para buscar ou validar textos em formato conhecido, porém de conteúdo variável.
  - Data e horário
  - Endereços IP ou MAC
  - e-mail
  - Dados entre $< tag >$ e $< /tag >$
  - Numero de telefone, RG e CPF.
- Como funciona?
- Como se utiliza?

Caracteres e meta-caracteres

Caracteres

- a b c ... z
- A B C ... Z
- 0 2 3 ... 9

Meta-caracteres

- $
- ?
- *

Meta-caracteres

^	Circunflexo	Representa o começo da linha
$	Cifrão	Representa o fim da linha
[abc]	Lista	Casa as letras a, b ou c
[a-d]	Lista	Casa qualquer letra entre a e d
[^abc]	Lista negada	Casa qualquer carácter exceto a,b e c
(esse|aquele)	Ou	Casa as strings *esse* ou *aquele*

Meta-caracteres

a{2}	Chaves	Casa *a* 2 vezes
a{2,4}	Chaves	Casa *a* de 2 a 4 vezes
a{2,}	chaves	Casa *a* no minimo duas vezes
a?	Opcional	Letra *a* é opcional
a*	Asterisco	Letra *a* zero ou mais vezes
a+	Mais	Letra *a* pelo menos uma vez
.	Ponto	Qualquer carácter uma vez
.*	Curinga	Qualquer coisa

grep

Uso: egrep [opções] *regex* arquivo1 [arquivo2] [arquivo3]

Exemplos:

egrep 'anna' names.txt	Tem *anna* em algum lugar
egrep '^anna' names.txt	Começa com *anna*
egrep 'anna$' names.txt	Termina com *anna*
egrep '^anna$' names.txt	Começa e termina com *anna*
egrep '^.?anna$' names.txt	Algum carácter seguido de *anna*
egrep 'an{1,2}a$' names.txt	Tem *anna* ou *ana*

Validação de ips

Um endereço de IP é composto de:

- 4 números
- Separados por 3 pontos
- Num intervalo de 0 até 255

	Correto	Incorreto
	0.0.0.0	0.0..0
Exemplos:	127.0.0.1	.127.0.10
	192.168.0.1	192.168.0.1.2
	255.255.255.255	256.265.255.266

Procurando por endereços de IP dentro de um arquivo

egrep '([0-9]{1,3}\.){3,3}[0-9]{1,3}' ips.txt

Entendendo a *regex* acima:

- '[0-9]' Casa qualquer carácter 0, 1, 2 ... 9
- '[0-9]{1,3}' Casa qualquer numero de 0 até 999
- '([0-9]{1,3}\.)' 0 até 999 seguindo de .
- '([0-9]{1,3}\.){3,3}' Acima repetido 3x
- '([0-9]{1,3}\.){3,3}[0-9]{1,3}' Acima seguido de 0 até 999

Qual o problema desta *regex* ?

Procurando por endereços de IP dentro de um arquivo

egrep '([0-9]{1,3}\.){3,3}[0-9]{1,3}' ips.txt

Entendendo a *regex* acima:

- '[0-9]' Casa qualquer carácter 0, 1, 2 ... 9
- '[0-9]{1,3}' Casa qualquer numero de 0 até 999
- '([0-9]{1,3}\.)' 0 até 999 seguindo de .
- '([0-9]{1,3}\.){3,3}' Acima repetido 3x
- '([0-9]{1,3}\.){3,3}[0-9]{1,3}' Acima seguido de 0 até 999

Qual o problema desta *regex* ? *Regex* correta: ^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5]).){3}
([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$

Um teste

Acesse `https://github.com/h3nnn4n/regex_examples` e abra o arquivo seu_nome.txt

Cole no terminal um por um e veja o resultado

Valendo uma camiseta **suprema** do Colmeia!!!

Conte quantas palavras existem no arquivo **names.txt**...

que começam com 2 vogais, tem ao todo 7 letras e não terminam em consoante.