Bash – 1.0

Tiago H31nr1ch
heinrichtx@gmail.com

03/2020

1 Instalação (Vídeo VirtualBox)

Para as atividades futuro é necessário a instalação do VirtualBox (here) que permite a instalação do ubuntu server (here) (recomendo versão 19). O ubuntu server permite a execução dos comandos de uma forma prática. Lembre de configurar a rede para o modo bridged, caso queira acessar a máquina virtual (VM) com o SSH. Para obter o endereço da VM, basta logar no *host* e buscar com o comando **ip a**.

Recomendo utilizar o *Windows Power Shell* (here) e o *Open SSH* que facilita na hora de fazer as atividades já que a máquina virtual pode ser acessada pelo SSH, não necessita de nenhum download externo basta acessar Settings -> App -> Optional Features -> selecionar instalar Open SSH (here).

2 Bash (Vídeo aula 1)

O shell Bash é comumente usado de maneira interativa, sistemas operacionais baseados em Unix usam o *Bash* como shell padrão (Linux e MacOS). O **echo** será utilizado para imprimir conteúdo de variáveis ou impressão de mensagens para o usuário (outra opção válida é utilizar o **printf "ola"**). Uma varável no Bash será declarada com o sinal $ no seu início, então para efetuar a leitura de uma informação do usuário basta utilizar o comando **read** seguido da variável ao qual a informação será atribuída.

```
1 echo "Whats your name?"
2 read name
3 # read input
4 echo "$name"
5 # print $name
```

A primeira linha de um *script* deve conter a sequência de caracteres **#!** (conhecidos como *shebang*) e sendo responsável por indicar que o arquivo é um "executável" e como deve ser executado. O *shebang* instrui o sistema operacional a executar */bin/bash* para o *scritp*. Por exemplo, ao executar **./script.sh** o sistema operacional irá receber **/bin/bash script.sh** (para que essa execução sejá possível é necessário a permissão de execução, lembrem do **chmod +x**).

```
#!/bin/bash
echo "Whats your name?"
read name
echo "$name"
```

No caso de uma variável alguns opções podem ser utilizadas para o armazenamento de valores. Possibilitando operações matemáticas como apresentado a soma com **expr**.

```
first_val="Tiago"
second_val=1
third_val=2
fourth_val=false

echo $first_val
printf "Output: $first_val"

expr $second_val + $third_val
```

Um *script* permite a otimização de rotinas, como a atividade realizada na última aula. Ao qual um *script* é responsável por gerar uma árvore de diretório, arquivos e links.

```
echo 'Exercise Build'
mkdir tree0
cd tree0
mkdir folder1 folder2 folder3 folder4
touch file1.txt file2.txt file3.txt system.log
chmod 000 file*.txt
chmod 777 system.log
chmod a+=rw folder1 folder2 folder3 folder4
cd ..
mkdir tree1
cd tree1
mkdir .folder-tree1 .folder-tree2
touch .file-a.txt
cd .folder-tree1
mkdir random tmp
cd tmp
touch README.md
echo -e '\n Ola, td bem?' > README.md
cd ../random/
mkdir random-1 random-2
```

```
21 touch random-1/remove-me.txt random-1/remove-me2.txt random-1/remove-me3.txt
22 cd random-2
23 mkdir change-me1 change-me2 change-me3
24 cd change-me1
25 touch link1.txt ../change-me2/link2.txt
26 echo 'jdsak daskjdsakjjkdsakdajsdasjk kjdasjk ads dsadsak' > link1.txt
27 echo 'skjas dsakjda sda dsa d sa ds ds sa ds  sa das' > ↩
       ../change-me2/link2.txt
28 ln link1.txt ../change-me3/some-file1.txt
29 ln ../change-me2/link2.txt ../change-me3/some-file2.txt
30 echo 'Done'
```

A descoberta de caminhos através do comando **which** ou **pwd** pode ser aplicado com um conjunto de operações para descoberta de aplicações otimizando processos de verificação.

```
1 dir='/home/'
2 cd $dir
3
4 app='gcc'
5
6 path=$(which $app)
7 current_path=$(pwd)
8
9 echo $path $current_path
```

Segue um exemplo prático para buscar informações no diretório */etc/*. Na linha 3 é apresentado todo o conteúdo do diretório (um por linha). Na linha 5 é realizado uma busca por todas as aplicações encontradas dentro da lista do diretório com permissão de usuário de leitura, escrita e execução. Na linha 7 é listado todas as linhas que possuem a *string sys*. Na linha 9 é apresentado todos os arquivos com o formato *.config*.

```
1 cd /etc/ ; values=$(ls -l)
2
3 printf "$values\n" # ou use echo -n "$values" para reconhecer o \n
4
5 printf "$values\n" | awk '/.rwx...../{print $0}'
6
7 printf "$values\n" | grep 'sys'
8
9 printf "%s\n" *.conf
```

Um exemplo básico de operações com processos é:

```
1 sleep 3
2
3 sleep 10 &
4 ps aux | grep sleep
5 jobs
```

```
fg
```

A repetição permite a otimização de rotinas repetitivas, como a criação de N diretórios e arquivos.

```
for i in {1..10}; do
    mkdir foulder-$i;
done

for i in {3,5,7,9}; do
    touch file-$i;
done
```

3 Estruturas de controle

Operadores && (and) e || (or) são permitidos no ambiente, permitindo a verificação de conteúdo, como é apresentando a seguir. A utilização de *if* no lugar de operadores resulta no mesmo resultado, MAS operadores tem um desempenho superior (bash é contrário de muitas linguagens como C, em que a memória é explicitamente alocada para estruturas e variáveis).

```
cd my_directory && ls || echo "No such directory"
```

Operadores para Arquivos

- **-e "$file"** Retorna true se o arquivo existir
- **-d "$file"** Retorna true se o arquivo existe e é um diretório
- **-f "$file"** Retorna true se o arquivo existe e é um arquivo regular
- **-h "$file"** Retorna true se o arquivo existir e for um link simbólico

Operadores para comparação de Strings

- **-z "$str"** Verdadeiro se o comprimento da string for zero
- **-n "$str"** Verdadeiro se o comprimento da sequência for diferente de zero
- **"$str1" = "$str2"** Verdadeiro se a string *$str1* for igual à string *$str2*. Não é o melhor para números inteiros (Pode funcionar, mas será inconsistente)
- **"$str1" != "$str2"** Verdadeiro se as cadeias não forem iguais

Operadores para comparação de Inteiros

- **“$int1” -eq “$int2”** Verdadeiro se os números inteiros forem iguais
- **“$int1” -ne “$int2”** Verdadeiro se os números inteiros não forem iguais
- **“$int1” -gt “$int2”** Verdadeiro se int1 for maior que int2
- **“$int1” -qe “$int2”** Verdadeiro se int1 for maior ou igual a int2
- **“$int1” -lt “$int2”** Verdadeiro se int1 for menor que int2
- **“$int1” -le “$int2”** Verdadeiro se int1 for menor ou igual a int2

Para realizar comparações utilizando os operadores:

```
if [[ 100 -eq 100 ]]; then
    echo "Equal"
fi

if [[ 100 -ne 200 ]]; then
    echo "Not Equal"
fi

if [[ 100 -le 200 ]]; then
    echo "100 < 200"
fi
```

A combinação de comandos para a busca de valores é possível, como:

```
echo -n "aaaa ola\ndsa\nd\na\nola\n" > a.txt
if grep "ola" a.txt; then
    echo "ola was found"
else
    echo "ola was not found"
fi
```

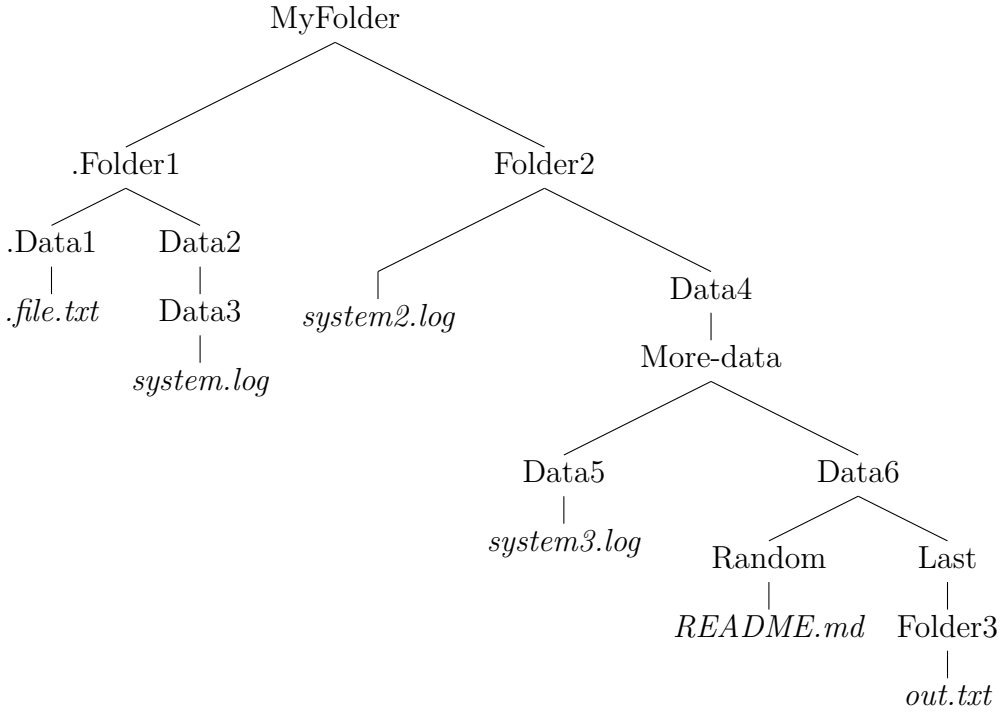
Por fim, existe a possibilidade de realizar um conjunto de operações, como:

```
if [[ $1 -eq 1 ]]; then
    echo "1 was passed in the first parameter"
elif [[ $1 -gt 2 ]]; then
    echo "2 was not passed in the first parameter"
else
    echo "The first parameter was not 1 and is not more than 2."
fi
```

4 Atividade para presença

Para a entrega só é necessário o envio de um *script* com as soluções.

1. Qual a diferença entre o Bash e o *Bourne Shell* (sh) ?
2. Faça a leitura de um nome e crie o diretório com o respectivo.
3. Faça um **for** para criar os arquivos **file-N** com as seguintes sequencias **{1..3}**, **{a,b,c}** e **{1,3,300}**.
4. Faça um *scritp* para construir a seguinte árvore no sistema:



5. Como devo fazer a comparação dos seguintes valores (utilize o **if**)?
   - $2 > 1$: True
   - $2 > 4$: False
   - $1 == 1$: True
   - "String"! = "string"