Shell Scripting

Introdução a BASH & BASH *scripting*

PEDRO MARTINS

January 1, 2018

Contents

1 Shell Scripting

1.1 Exercise 1 - Command Overview

- **man** : Documentação dos comandos
- **ls** : Listar ficheiros de uma pasta
- **mkdir** : Criar uma pasta
- **pwd** : Caminho absoluto do diretório corrente
- **rm** : Remover ficheiros
- **mv** : Renomear ficheiros ou mover ficheiros/pastas entre pastas
- **cat** : Imprimir um ficheiro para o stdout
- **echo** : Imprimir para o stdout uma mensagem
- **less** : paginar um ficheiro (não mostra o texto literal)
- **head** : mostrar as primeiras 10 linhas de um ficheiro
- **tail** : mostrar as ultimas 10 linhas de m ficheiro
- **cp** : copiar ficheiros
- **diff** : mostrar as diferenças linha a linha entre dois ficheiros
- **wc** : contar linhas, palavras e caracteres de um ficheiro
- **sort** : ordenar ficheiros
- **grep** : pesquisa de padroes em ficheiros
- **sed** : transformacoes de texto
- **tr** : substituir, modificar ou apagar caracteres do stdin e imprimir no stdout
- **cut** : imprimir partes de um ficheiro para o stdout
- **paste** : imprimir linhas de um ficheiro separadas por tabs para o stdout

- **tee** : Redireciona para o nome do ficheiro passado como argumento e para o stdout

1.2 Exercise 2 - Redirect input and output

1.2.1 1

> : redirecionar o output do comando anterior do stdout para um ficheiro
>> : append do output do comando anterior do stdout para um ficheiro

1.2.2 2

2> : redireciona o stderror para um ficheiro

1.2.3 3

| : redireciona o stdout de um comando para o stdin do comando seguinte

1.2.4 4

2>&1 : redireciona o stderror para o stdout
1>&2 : redireciona o stdout para o stderror

1.3 Exercise 3 - Using special characters

1.3.1 1

touch : criar ficheiros caso o ficheiro não exista. Alterar a data de modificação caso o ficheiro exista
a* : [REGEX] Lista todos os ficheiros que o primeiro caracter seja um a, independentemente do número de ficheiros
a? : [REGEX] Lista todos os ficheiros começados por a e com mais 1 caracter
* : [REGEX] Lista qualquer ficheiro independentemente do numero de caracteres

1.3.2 2

[ac] : [REGEX] Lista os ficheiros com os caracteres entre []
[a-c] : [REGEX] Lista os ficheiros com os caracteres entre a e c
[ab]* : [REGEX] Lista os ficheiros com os caracteres {a, b} independentemente do número de caracteres

1.3.3 3

o \ antes de um caracter especial desativa as capacidades especiais do stdout

a* : [REGEX] Lista todos os ficheiros começados por a independentemente do número de caracteres
a* : Lista o ficheiro com o nome a*
a? : [REGEX] Lista todos os ficheiros começados por a e com mais um caracter
a\? : Lista o ficheiro com o nome a?
a\[: Lista o ficheiro com o nome a[
a\\ : Lista o ficheiro com o nome a

1.3.4 4

Usando `''` ou `""` podemos desativar o significado de caracteres especiais
`a*` : [REGEX] Lista todos os ficheiros começados por a independentemente do número de caracteres
`'a*'` : Seleciona o ficheiro a*
`"a*"` : Seleciona o ficheiro a*

1.4 Exercise 4 - Declaring and using variables

1.4.1 1

<variable name>=…. : Atribuição de variáveis em bash. Não deve ter espaço entre o nome da variável e a atribuição
`$<variable name>` : lê o valor da variável (em bash existe diferença entre atribuir um valor a uma variável e ler o valor da variável). Pode se atribuir nome de ficheiros e usar REGEX (p.e. z=a*)
`${<variable name>}` : lê o valor da variável (em bash existe diferença entre atribuir m valor a uma variável e ler o valor da variável)
`${<variable name>}<etc>` : Concatena o valor da variável com o que está à frente ()

1.4.2 2

- `$<variable name>` : Acede ao valor da variável
- `"$<varibale name>"` : Acede ao valor da variável (não aplica quaisquer caracteres especiais). P.e. se v=a, `"$v"` *será igual a a* em vez de todos os ficheiros começados por a com mais um caracter adicional
- `'$<variable name>'` : Ignora a leitura da variável e de um possível REGEX, devolvendo `$<variable name>`

1.4.3 3

- `${<variable name>:start:numero de caracteres}` : trata a variável como string, criando uma substring começando no caracter start com o numero de caracteres especificado. Pode ter espaços entre os :

- `${<variable name\>/<search substring\>/<replace substring>}` : Procura uma substring na variable name e substitui por outra substring indicada

1.5 Exercise 5 - Declaring and using functions

1.5.1 1

Para declarar uma função:

```
1 <nome_da_funcao>()
2 {
3     # corpo da função
4 }
```

1.5.2 2

`$#` : Número de argumentos de uma função
`$1` : Primeiro argumento
`$2` : Segundo argumento
`$*` : Todos os argumentos - Ignora sequencias de white space dentro das aspas na passagem de argumentos da bash
`$@` : Todos os argumentos - Ignora sequencias de white space dentro das aspas na passagem de argumentos da bash
`"$*"` : Todos os argumentos - Preserva a forma dos argumentos passados entre aspas (i.e., o white space)
`"$@"` : Todos os argumentos - Preserva a forma dos argumentos passados entre aspas (i.e., o white space)

1.6 Exercise 6

1.6.1 1, 2 e 3

- `{ .......\}` : Agrupar commandos (pode ser redirecionado o stdout usando | ou >). A lista de comandos é executada na mesma instância da bash em que é chamada (contexto global de execução, com variáveis globais)

- (…….) : Agrupar comandos (pode ser redirecionado o stdout usando | ou >). O grupo de comandos é executado noutra instância da bash (contexto próprio de execução, com variáveis locais)

1.7 Exercise 7

1.7.1 1

$? : Valor de retorno de um comando (semelhante a C/C++). Se for '1' existe um erro na execução do comando. Se for '0' está tudo bem

1.7.2 2

```
echo -e : Faz parse de códigos de cores

"e\33m ... \e[0m" : Código de cores que define a cor de sucesso
"e\31m ... \e[0m" : Código de cores que define a cor de erro
```

Estrutura de um if:

```
if <cond>
then
    <statment>
else
    <statment>
fi
```

1.7.3 3

Os parentesis retos na condição do if (p.e. `if [ -f $1]`) que chamam a função test devem estar com pelo menos um espaço entre os outros caracteres

1.7.4 4

Os operadores têm de estar com pelo menos um espaço de intervalo
! : Operador not

1.7.5 5

&& : Operador and
|| : Operador or

1.8 Exercise 8 - The multiple choice case construction

1.8.1 1 Case stament

```
case <variavel para selecionar> in
    <cond1>) <statment 1>;;
    <cond2>) <statment 2>;;
    <cond3>) <statment 3>;;
    ....
esac
```

Onde:
- A pode ser $#, $* ou $1
- O ;; no final da <ação #> equivale ao fim da branch (break em C)
- O | permite a definição de uma várias alternativas (condições) para o mesmo case (e consequentemente ação)
- O * significa qualquer valor. Ao ser colocado em último permite selecionar todas as outras opções que ainda não forma cobertas (equivalente ao default em C)
Exercise 9 - The repetitive for contruction

1.8.2 1

A syntax de um for é:

```
for <variavel de iteracao> in <lista de objectos para iterar>
do
    <statment>
done
```

Onde <lista de objectos para iterar> podem ser ficheiros e/ou pastas e podem ser usados caracteres especiais como a*

1.8.3 2

1.9 10 - The repetitive while and until constructions

1.9.1 1

Estrutura de um while:

```
while [ <condicao de paragem> ]
do
    <statment>
    shift
done
```

Estrutura de um until

```
until [ <condição de paragem> ]
do
    <statment>
    shift
done
```

Onde a condição de paragem pode ser escrita como: `<variable> <condição de teste> <fim>`

As condições de teste podem ser:

```
-gt : greater than
-eq : equal
-lt : less than
-le : less or equal than
-ge : greater or equal than
```

`shift` é uma palavra equivalente ao continue em C

1.10 Exercise 11 - Script Files

2 1

O cabeçalho do ficheiro de script é:

```
#!/bin/bash
# The previous line (comment) tells the operating system that
#  this script is to be executed in bash
#
```

Condições usadas:

```
[ $# -ne 1 ] : número de argumentos diferente de 1
! [ -f $1 ] : O primeiro argumento dado não é um ficheiro
1>&2 - Redirecionar o stderror para o stdout
```

2.1 Exercise 12 - Bash supports both indexed and associative arrays

2.1.1 1

Os indices de um array não são continuos e não podem ser negativos

A declaração explicita dos arrays pode ser feita fazendo: `declare -a <array>[<idx>]=<value>`
Outras operações:

- **Atribuição**: `<array>[<idx>]=<valor>`
- **Leitura**: `${<array>[<idx]}`
- **Leitura de todos os elementos do array**: `${a[*]}`
- **Número de elementos do array**: `${#a[*]}`
- **Lista dos indices do array** : `${!a[*]}`

Os indices podem ser obtidos com expressões aritméticas
A iteração pelos indices é feita da mesma forma que em python
- **Iterar na lista de elementos**: `for <variavel> in ${<array>[*]}`
- **Iterar na lista de indices**: `for <variavel> in ${!a[*]}`

Exemplo de código para imprimir os indices e os elementos

```
for v in ${!a[*]}
do
    echo "a[$i] = ${a[$i]}"
done
```

2.1.2 2

A declaração de arrays associativos tem de ser feita de forma explicita `declare -A <array>`

- A **atribuição de valores** para um **array associativo**: `<array>["<key>"]=<value>`
- **Listar os elementos no array** : `${<array>[*]}`
- **Listar o número de elementos no array** : `${#<array>[*]}`
- **Listar os indices usados no array** : `${!<array>[*]}`

Exemplo de código para percorrer as keys e imprimir as keys e os values

```
for i in ${!arr[*]}
do
    echo "Key = $i | Value = ${arr[$i]}\"
done
```