CENTRO DE FORMAÇÃO
R.P. DE ALCOITÃO

GUIA DE LABORATÓRIO 2.5 - GREP E EXPRESSÕES REGULARES - RESOLUÇÃO DOS EXERCÍCIOS (ßeta)

EXERCÍCIOS PRÁTICOS

1. Considere o ficheiro dados.txt com o seguinte conteúdo:

```
1  Alberto 10
2  Armando 20 Alberto 30
3  30 Bruno
4  20 Alberto
5  10 alberto 20 armando
6  bruno 30
7  100 ALBERTO 20 Bernardo
8  ### FIM
```

Qual a saída dos seguintes comandos?

(a) `grep '10' dados.txt`

```
Alberto 10
10 alberto 20 armando
100 ALBERTO 20 Bernardo
```

(b) `grep -v '10' dados.txt`

```
Armando 20 Alberto 30
30 Bruno
20 Alberto
bruno 30
### FIM
```

(c) `grep -i -n  '0 a' dados.txt`

NOTA: *`'0'` (zero) seguido de um espaço seguido de `'a'`*

```
2:Armando 20 Alberto 30
4:20 Alberto
5:10 alberto 20 armando
7:100 ALBERTO 20 Bernardo
```

(d) `grep -E '^A' dados.txt`

```
Alberto 10
Armando 20 Alberto 30
```

(e) `grep -E '^[[:alpha:]]' dados.txt`

```
Alberto 10
Armando 20 Alberto 30
bruno 30
```

(f) `grep -vE '[[:digit:]]+' dados.txt`

```
### FIM
```

(g) `grep -E '20 (B|Al)' dados.txt`

```
Armando 20 Alberto 30
20 Alberto
100 ALBERTO 20 Bernardo
```

(h) `grep -E '[[:alpha:]]{7}' dados.txt | grep -v '20'`

```
Alberto 10
```

2. Escreva expressões regulares apropriadas para utilizar com o **grep** para:

2.1 `^[aeiouAEIOU]`

2.2 `[aeiouAEIOU0-9]$`

2.3 `^([aeiouAEIOU].*[^0-9]|[^aeiouAEIOU].*[0-9])$`

2.4

Para palavras contendo apenas letras (não acentuadas):

```
CONS="[bcdfg...z]"   # ou seja, todas as consoantes
\b$CONS*[aeiou]$CONS*[aeiou]$CONS*\b
```

Para palavras entendidas como sequências de caracteres sem espaços (não era pedido):

```
\b[^aeiou ]*[aeiou][^aeiou ]*[aeiou][^aeiou ]*\b
```

2.5 `//.*$`

NOTA: Também se aceita `^//.*$`

2.6 `'^$'`

2.7 `'^[[:blank:]]*$'`

$ `grep -vE '^[[:blank:]]*$' ficheiro`

2.8 `'[[:lower:]]([[:digit:]]{2}|[[:digit:]]{4})\b'`

2.9 `'.+[[:lower:]]([[:digit:]]{2}|[[:digit:]]{4})\b.+'`

2.10

Primeiro caso: `Linux|UNIX`

$ `grep (Linux|UNIX)`

Segundo caso:

$ `grep Linux | grep UNIX`

ou, utilizando apenas expressões regulares, sem utilizar pipes,

`(Linux.*UNIX)|(UNIX.*Linux)`

2.11 Não delimitado: `'[0-9]{1,3}'`

Delimitado por espaçamento: `'\s[0-9]{1,3}\s'`

Delimitada por início/fim de palavra: `'\b[0-9]{1,3}\b'`

NOTA: ver considerações a propósito da delimitação no execício para filtrar pseudo-endereços de email

2.12 Não delimitado: `'[0-9]{1,3}\.'`

Delimitado por espaçamento: `'\s[0-9]{1,3}\.'`

Delimitada por início/fim de palavra: `'\b[0-9]{1,3}\.'`

2.13

```
DIG_IP='(25[0-5]|2[0-4][0-9]|1[0-9][0-9]|[1-9][0-9]|[1-9])'
```

E depois fazemos (assumindo que se pretende delimitar a expressão:

```
grep -E "\b$DIG_IP\.$DIG_IP\.$DIG_IP\.$DIG_IP\b"
```

2.14 O melhor é dividir em duas partes:

```
PARTE_ESQ='[[:alpha:]][[:alnum:]]*(_[[:alnum:]]+)?'
```

```
PARTE_DIR='[[:alpha:]][[:alnum:]]*\.[[:alpha:]][[:alnum:]]*'
```

E depois invocar o **grep** deste modo:

```
$ grep -wE "$PARTE_ESQ@$PARTE_DIR"
```

A opção `-w` faz com expressões como `1xpto@mail.com` não sejam seleccionadas. Uma vez que a expressão regular `PARTE_ESQ` deve começar com um caractere alfabético, o `-w` obriga a que esta seja delimitada por um caractere que não faça parte de uma palavra - um dígito é considerado com um caractere passível de fazer parte de uma palavra.

Mas o `-w` não impede a selecção de linhas que contenham algo como `xpto@mail.com@bla.pt` . A expressão regular `PARTE_DIR` indica que esta termina num alfanumérico e o `-w` força que a seguir venha um caractere não constituinte de uma palavra. Ora, tudo o que não é alfanumérico é considerado não constituinte de palavra e assim o `@` delimita a expressão. Se pretendermos que a expressão esteja completamente isolada por caracteres brancos ou pelo início/fim de linha devemos alterar o comando para:

```
$ grep -wE "(^|[[:blank:]])$PARTE_ESQ@$PARTE_DIR([[:blank:]]|$)"
```