Capítulo 2

===Exercício 1===

Descubra o resultado da execução dos comandos seguintes (primeiramente tente identificar a resposta sem o uso de computador, em seguida use-o para certificar-se do resultado):

A) `sed 's#UNIX#unix#' quequeisso`

Troca o 1º `UNIX` de cada linha por `unix`

B) `sed '1,2s/[A-Z]//' quequeisso`

Nas linhas `1` e `2,` apaga a primeira maiúscula de cada linha

C) `sed '1,2s/[a-z]//g' quequeisso`

Apaga todas as letras minúsculas da 1ª e 2ª linha

D) `sed 's/ .*//' quequeisso`

Apaga tudo após o primeiro espaço em branco

E) `sed -n '/UNIX/p' quequeisso`

Só imprime as linhas que têm a cadeia `UNIX`

F) `sed '/UNIX/d' quequeisso`

Deleta as linhas que têm a cadeia `UNIX`

G) `expr 15 / 2 * 5`

Se houver algum arquivo no diretório corrente o `*` vai expandir para seu(s) nome(s) o que ocasionará erro

H) `expr 15 \* 5 / 2`

Executará essa operação que terá com resultado `37.` No entanto, essa mesma conta poderá ser feita de forma muito mais veloz com:

```
echo $((15*5/2))
```

I) `grep '^(P|S)' quequeisso`

Não casará com nada, pois sem a opção `-E` os parênteses e a barra vertical não são metacaracteres, são literais. Só haveria casamento se houvesse alguma linha começando `(^)` por `(E|C)`

J) `grep -E '^(E|C)' quequeisso`

Exibe todas as linhas que começam por `E` ou por `C`

K) `grep -E -v '^(E|C)' quequeisso | wc -l`

Deleta as linhas começadas por `E` ou por `C` e conta as restantes

L) `grep -F -l ou *`

Procura em todos os arquivos do diretório `(*)` a cadeia ou e lista os nomes dos arquivos que tenham essa cadeia. A opção `-F` foi para acelerar, já que não há expressão regular.

M) `expr length "C qui sabe"`

Mede o tamanho da cadeia, porém prefiro fazer:

```
$ Var="C qui sabe"
$ echo ${#Var}
```

É mais de 100 vezes mais rápido.

===Exercício 2===

Como faço para pegar o tamanho dos arquivos e seus nomes (**dica**: veja a opção `-s` do comando `tr`)

Com o que aprendemos até agora, a solução mais simples é:

```
$ ls -l | tr -s ' ' | cut -f 5,9 -d ' '
```

Ou, para melhorar a apresentação:

```
$ ls -l | tr -s ' ' '\t' | cut -f 5,9
```

Nessa última, a opção `-s` (*squeeze*) compactou os espaços e transformou-os em `<TAB>`

===Exercício 3===

Considere um arquivo, chamado `numeros` com o seguinte conteúdo:

```
1234567890
0987654321
1234554321
9876556789
```

Escreva o comando que gerará a seguinte saída:

```
13579
08642
13542
97568
```

```
$ cut -c 1,3,5,7,9 numeros
```

===Exercício 4===

Considere o arquivo `arq.txt` que possui apenas uma linha com o seguinte conteúdo:

```
$ cat arq.txt
MMMMMAMMMMMMMMO ACEMMMMMMMMMTAMMMMMMMM!
```

Qual será o conteúdo do arquivo `arq1.txt` ao final da execução do seguinte comando?

```
$ tr -s M R < arq.txt > arq1.txt
```
A opção `-s` desse comando compacta os `M` transformando os que restaram em `R`, produzindo a cadeia `RARO ACERTAR!` que é gravada no arquivo `arq1.txt`

===Exercício 5===

Se você fizer:

```
$ seq -w 30 | paste -d: - - - - -
```

Obterá:

```
01:02:03:04:05
06:07:08:09:10
11:12:13:14:15
16:17:18:19:20
21:22:23:24:25
26:27:28:29:30
```

Após esse, que comando devo executar para que a saída seja:

```
01:02:03=04=05
06:07:08=09=10
11:12:13=14=15
16:17:18=19=20
21:22:23=24=25
26:27:28=29=30
```

Isto é o a linha de comandos que vimos acima, deve ser passada (por pipe) para outro comando que trocará os dois pontos (:) pelo sinal de igual (=), mas só a partir do terceiro.

Comandos válidos:

Somente um `sed`

A resposta pode ser a linha completa

```
$ seq -w 30 | paste -d: - - - - - | sed 's/:/=/3g'
```

Ou somente o `sed` que foi solicitado como complemento desta linha:

```
      ...| sed 's/:/=/3g'
```

===Exercício 6===

Explique porque o comando a seguir deu esta saída:

```
$ sed 'ss\ssusg' <<< sussurar
uuuuurar
```

O comando `s` (*substitutute*) do `sed` aceita o caractere que segue o `s` como sendo o separador dos argumentos e, neste caso, usei a letra `s` como tal. Então o comando serve para substituir `(s)` todos `(g)` os caracteres `s (\s)` por caracteres `u.`

===Exercício 7===

Sabendo que os campos de `/etc/passwd` são separados por dois pontos e o leiaute de seus 4 primeiros campos é:

```
UserName:x:UID:GID: ...
```

Bolar uma forma de listar o *UserName* de todos os registros que tiverem o mesmo número de UID e de GID.

Observação:

UID - *User Id* - Número com a identificação do usuário;

GID - *Group Id* - Número com a identificação do grupo do usuário

```
$ grep -E '^.*:([0-9]+):\1:' /etc/passwd | grep -Eo '^[^:]+'
```

Explicação:

```
:([0-9]+):\1:
```

Casa com números igual e separados por 2 pontos (:). Experimente fazer:

```
$ grep -E ':([0-9]+):\1:' /etc/passwd
```

Beleza, mostrou um monte de registro com número duplicado, só para provar que a *Expressão Regular* está certa, mas eu pedi o *login name* dos caras. Como o pedido foi *login*, fica mais estético começar dele (embora desnecessário). Então o `grep` seria:

```
$ grep -E '^.*:([0-9]+):\1:' /etc/passwd
```

Como o login name já está na resposta, vamos jogar fora o que não interessa:

```
$ grep -E '^.*:([0-9]+):\1:' /etc/passwd | grep -Eo '^[^:]+'
```

Nesse último `grep,` a opção `-o` só vai mostrar o que casou e assim sendo, o `^[^:]+` vai casando, a partir do início `(^),` com tudo que não seja dois pontos `[^:]+,` ou seja, o *login name*.

===Exercício 8===

Como posso listar somente o 23º registro do arquivo `/etc/passwd?`

Comando válido: `grep`

O `grep` com a opção `-n` mostra os números das linhas que casaram com o escopo do `grep.` Como o metacaractere ponto `(.)` é uma *Expressão Regular* que casa com tudo, podemos numerar todas as linhas pesquisando por ela, veja:

```
$ grep -n . /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
2:daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
```

Ou seja, recebemos o número da linha separado do registro por um dois pontos `(:)` então basta procurar o 23º registro no início `(^)` da linha:

```
$ grep -n . /etc/passwd | grep ^23:
```