Capítulo 1

===Exercício 1===

O que aconteceria na execução dessas sequências de comandos?

A) `ls | wc -l`

O `ls` lista os arquivos e `wc -l` conta quantas linhas foram geradas, isto é a quantidade de arquivos

B) `mail procara << !`

Tudo que estiver entre essa linha e linha que contiver unicamente um ponto de exclamação (!), servirá de entrada para o comando mail

C) `cat quequeisso | tee qqisso`

(Que seria mais veloz se fosse assim: `tee qqisso < quequeisso`)

Coloca o conteúdo do arquivo `quequeisso` na tela e, simultaneamente, no arquivo `qqisso`

D) `(cd ; pwd)`

Os parênteses criarão um *subshell* que irá para o diretório *home* `(cd)` e listará o nome (caminho completo) desse diretório. Ao seu fim, voltará à pasta anterior, pois o filho foi para o *home*, mas o pai nunca saiu de sua posição

E) `mail procara < mala`

Manda um e-mail `procara` com o conteúdo do arquivo `mala`

F) `ls -l NuncaVi >> /tmp/$$ 2>> /tmp/x$$`

Gera uma listagem longa do arquivo `NuncaVi` e anexa-a ao fim do arquivo `/tmp/$$.`

Se houver algum erro na execução, a mensagem de erro será anexada ao `/tmp/x$$.`

OBS: `$$` é a variável que contém o número do processo (PID) do *Shell* que o está interpretando

G) `ls -l > /tmp/$$ 2> /tmp/x$`$

Manda a saída do `ls -l` para o arquivo `/tmp/$$` e os possíveis erros para `/tmp/x$$`

OBS1: os dois arquivos perderão o conteúdo anterior

OBS2: Veja a obervação do exercício anterior quanto à variável `$$`

H) `echo Nome do Sistema: uname -n`

Será escrito na tela:

`Nome do Sistema: uname -n`

Porque o interpretador lê da esquerda para a direita e o primeiro comando lido é o `echo`

I) `echo Nome do Sistema: `uname -n``

Será escrito na tela:

`Nome do Sistema: <O NOME DO SEU COMPUTADOR>`

As crases (acento grave) servem para priorizar o comando `uname -n` que gerará `<O NOME DO SEU COMPUTADOR>.` No lugar das crases é mais comum acharmos `$(uname -n)`

===Exercício 2===

Qual comando deve ser empregado para:

A) Executar o programa `prog,` mandando a sua saída simultaneamente para a tela e para o arquivo `prog.log.`

```
$ prog | tee prog.log
```

B) Listar todos os arquivos começados pelas letras `a, b, c, d, e, h, i, j, k, x` e que não terminam com esses mesmos caracteres.

```
$ ls [abcdehijkx]*[!abcdehijkx]
```

ou:

```
$ echo [abcdehijkx]*[!abcdehijkx]
```

C) Escrever na tela do terminal:

As seguintes pessoas estão logadas: <relação gerada pelo comando `who`>

```
$ echo As seguintes pessoas estão logadas: $(who)
```

ou:

```
$ echo As seguintes pessoas estão logadas: `who`
```

===Exercício 3===

Se eu fizesse:

```
$ cat arq > arq1
```

Qual seria o resultado?

Seria o mesmo que fazer:

```
$ cp arq arq1
```

Isto é, copiaria o conteúdo de `arq` para `arq1`

===Exercício 4===

O que aconteceria se eu fizesse (Cuidado com a casca de banana!):

```
$ cat arq > arq
```

Escolha uma:

A) Duplicaria o conteúdo de arq

B) Daria erro e o arquivo arq permaneceria inalterado

C) O arquivo arq seria apagado

D) Metade de arq seria duplicada no seu início e a outra metade no fim

E) Nenhuma das respostas acima

Quando o *Shell* vê uma linha de comandos, a primeira coisa que ele faz, são os redirecionamentos, assim sendo, antes de executar o comando `cat,` o conteúdo de `arq` seria apagado.

===Exercício 5===

Esse é para quem gosta de pensar e pesquisar. O comando:

```
$ echo '2.37*3.421' | {
    Res=$(bc)
    echo $Res
}
echo -$Res-
8.107
--
```

Como você viu, mandou dois números reais para a calculadora do *bash* `(bc)` que fez as contas e mandou o resultado. Isso é muito legal, mas o pipe `(|)` criou um *subshell* que quando morreu, levou junto com ele o valor da variável `$Res.`

Aí vem a dolorosa pergunta: como posso fazer o mesmo sem criar um *subshell* e sem perder o conteúdo da variável?

Para fazer o solicitado, basta redirecionar via *here strings* `(<<<)` a conta a ser feita para a entrada do bloco formado pelas chaves `({}),` ficando:

```
{
    Res=$(bc)
    echo $Res
} <<< '2.37*3.421'
echo -$Res-
8.107
-8.107-
```