Funções em Shell

São um bloco de comandos que, geralmente, recebem dados, fazem a manipulação destes através das instruções e devolvem algum resultado. Às vezes não retornam nada, mas fazem algum tipo de mudança no seu script, nem que seja uma simples exibição de texto. Repetimos muitos comandos e instruções. Escrevendo, definindo e salvando as funções de maneira que estas sejam bem simples e executem uma tarefa bem específica, é uma boa prática para evitar a repetição na escrita de código, pois podemos sempre reutilizar estas funções.

Forma 1

```
funcao(){
    #instruções
}
```

Forma 2

```
function funcao(){
    #instruções
}
```

Funções em Shell

Para chamar a função, basta repetir o nome dela no local desejado.

```
MinhaFuncao(){
    echo "Essa é minha função"
}

MinhaFuncao # saída: Essa é minha função
```

Não se deve usar nomes reservados, nem acentos, traços,... somente maiúscula, minúscula, misto, números e underlines.

Passando Parâmetros Para Funções em Shell

Lembra que nós falamos sobre variáveis especiais ($1 , $2, ...), então, elas representarão os parâmetros para função.

```
MinhaFuncao(){

        echo "Desenvolvo em $2 $1"
}

MinhaFuncao $1 $2
```

```
$ ./meuscript.sh Shell Script # saída: Desenvolvo em Script Shell
```

A palavra Shell foi alocada na variável $1 e Script na variável $2, perceba que a chamada no echo da função foi invertida, logo ficou Script Shell em vez de Shell Script (...)

Passando Parâmetros Para Funções em Shell

Assim como foi explicado sobre variáveis especiais
se você quiser exibir tudo, usa-se $@

```
MinhaFuncao(){

        echo "Todos os parâmetros que você passou : $@"
    }
```

./meuscript.sh Shell Script Bash
saída: Todos os parâmetros que você passou : Shell Script Bash

Lembrando que pra que tudo seja exibido, você precisa chamar dentro do script na chamada da função

MinhaFuncao $1 $2 $3
ou
MinhaFuncao $@

$# , conta os parâmetros foram passados,
$? , dentro da função informa se houve erro ou não, se a saída dessa variável for diferente de 0(zero), houve erro.

Ferramentas para Funções

I. O comando return informa o fim da função
II. Variáveis globais e locais

```
MinhaFuncao(){
        echo "Isso será exibido"
        return
        echo "Isso não será exibido, pois foi após o return"
}

MinhaFuncao # saída: Isso será exibido
```

Obs.: Mas para exibir a Global, terá que chamar o nome da função no script.

Se você criar uma variável dentro de uma função, mesmo chamado ela fora da função, o valor da variável será exibido, pois ela será tratada como variável GLOBAL, caso tente imprimi-la, para que o valor de uma variável seja exibido somente dentro da função, precisamos usar o comando local antes da variável para que não seja exibido.

Ferramentas para Funções

```
MinhaFuncao(){

        VariavelGlobal="Será exibida fora da Função"
        local VariavelLocal="Só será exibida dentro da Função"
}

MinhaFuncao

echo $VariavelGlobal # saída: Será exibida fora da Função
echo $VariavelLocal  # sem saída.
```

Diferente da variável que você pode mudar o valor dela a qualquer momento, as CONSTANTES não muda de valor e para declarar uma constante, a sintaxe é a seguinte

```
declare -r CONSTANTE='sempre igual'
echo $CONSTANTE # saída: sempre igual
```

Se tentar colocar um outro valor pra constante, gera erro

```
CONSTANTE="Mude" ; echo $CONSTANTE
# saída: a variável permite somente leitura
```

Ferramentas para Funções

Para apagar uma função, ou uma constante, também usa-se **unset** e o nome da função ou constante.

***Obs**: Constantes podem ser declaradas, mesmo fora das funções, abordamos aqui dentro, pois geralmente são usadas nas funções.*

```
unset MinhaFuncao
unset CONSTANTE
```

Mais uma observação, outro comando muito usado é o **source** , ele serve pra chamar arquivos externos, seria o ~~*include*~~, ~~*import*~~,... de outras linguagens, se você tiver um arquivo só com funções, você chamar esse aquivo no *script* que você estiver e depois apontar para uma função que está no arquivo externo

```
source arquivo_externo
```

É necessário frisar que se você criar uma função (ou *comando/alias*) com o nome de um comando que já existe, o Shell irá dar preferência para sua função, se você quiser usar o comando e a função com os mesmo nomes, use o comando builtin para chamar o comando do Shell em vez da sua função.

```
cd(){
    echo "Essa função tem o nome do comando cd e o parâmetro é $1"
}
cd meudiretorio/ # saída: Essa função tem o nome do comando cd e o parâmetro é meudiretorio/

    builtin cd Tarefas/ # irá entrar diretório.
    ls # lista os arquivos dentro do 'meudiretorio/'
```

Se você der um '**ls**' no terminal, você listará os arquivos fora do 'meudiretorio/' informado, mas se passar um '**ls**' no script, listará os arquivos dentro de 'meudiretorio/'

Pois o comando '**cd**' ele é um **built-in** , está embutido no Bash , para você saber se o comando está ou não embutido no Bash, use o comando como expansão de variável, **which [comando]**, se não retornar nada, ele é um **builtin**