

Shell Script [.sh]

Podemos utilizar outros tipos de shell para executar o arquivo **.sh**

Dependendo do Kernel e Distro Linux.

Mais comuns:

Bash, Ksh, Zsh, Csh, Fish, etc

Como instalar:

`sudo apt install nome-shell`

```
urubu100@DESKTOP-00I5LD3:~$ csh
The program 'csh' can be found in the following packages:
 * csh
 * tcsh
Try: sudo apt install <selected package>
urubu100@DESKTOP-00I5LD3:~$ sudo apt-get install csh
[sudo] password for urubu100:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  csh
0 upgraded, 1 newly installed, 0 to remove and 51 not upgraded.
Need to get 235 kB of archives.
After this operation, 367 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu xenial/universe amd64 csh amd64 20110502-2.1ubuntu1 [235 kB]
Fetched 235 kB in 1s (140 kB/s)
Selecting previously unselected package csh.
(Reading database ... 34541 files and directories currently installed.)
Preparing to unpack .../csh_20110502-2.1ubuntu1_amd64.deb ...
Unpacking csh (20110502-2.1ubuntu1) ...
Processing triggers for man-db (2.7.5-1) ...
Setting up csh (20110502-2.1ubuntu1) ...
update-alternatives: using /bin/bsd-csh to provide /bin/csh (csh) in auto mode
urubu100@DESKTOP-00I5LD3:~$
```

Digite **csh**, entre nesse shell e saia com **exit**

Alguns comandos são diferentes entre os shells

Embora tenhamos instalado o csh, iremos usar o bash

Para cada programa que você usa no computador, o shell cria um ambiente que possui variáveis de ambiente: **Globais** e **Locais**

#As variáveis globais são herdadas a todos os subprocessos do shell, inclusive para outros shells.

Exemplos:

PATH – lista diretórios de programas executáveis

USERNAME – nome do user logado

TERM – tipo de terminal em uso

HOME – diretório home do usuário ativo

UID – identificação do user ativo

RANDOM – gera números aleatórios

LANG – linguagem local – idioma

env – comando que mostra a variáveis de ambiente do terminal

printenv – comando que mostra a variáveis de ambiente do terminal

export – exporta as variáveis criadas

Usando o comando **env**, podemos ver as variáveis de ambiente globais

Observe que as variáveis de ambiente globais então em maiúscula

```
urubu100@DESKTOP-00I5LD3:~$ env
SHELL=/bin/bash
TERM=xterm-256color
WSLENV=
USER=urubu100
NAME=DESKTOP-00I5LD3
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;0
32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:
;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01
=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;
=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;
ng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm
.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli
gm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=0
ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
HOSTTYPE=x86_64
PATH=/home/urubu100/bin:/home/urubu100/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sb
pps/CanonicalGroupLimited.Ubuntu16.04onWindows_2020.1604.14.0_x64__79rhkp1fndgsc:/mnt/c/
hon27/Scripts:/mnt/c/Windows/System32:/mnt/c/Windows:/mnt/c/Windows/System32/wbem:/mnt/c
ProgramData/chocolatey/bin:/mnt/c/Program Files/Git/cmd:/mnt/c/Program Files (x86)/nodej
arise Miranda/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/Users/Marise Miranda/A
PWD=/home/urubu100
LANG=en_US.UTF-8
SHLVL=1
HOME=/home/urubu100
LOGNAME=urubu100
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
LESSOPEN=| /usr/bin/lesspipe %s
LESSCLOSE=/usr/bin/lesspipe %s %s
```

#Agora as variáveis locais são aquelas variáveis de ambiente do shell atual.

#Essas variáveis locais não são herdadas por outros locais ou shells.

Exemplos:

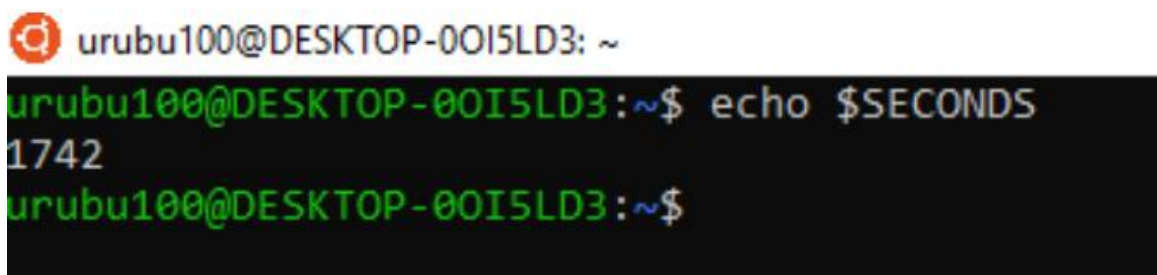
SHELL – indica qual o shell está sendo usado

SECONDS – tempo que o shell está ativo

Posso abrir um novo bash para que você perceba como funciona as variáveis de ambiente locais.

Abra o terminal e execute:

`echo $SECONDS`



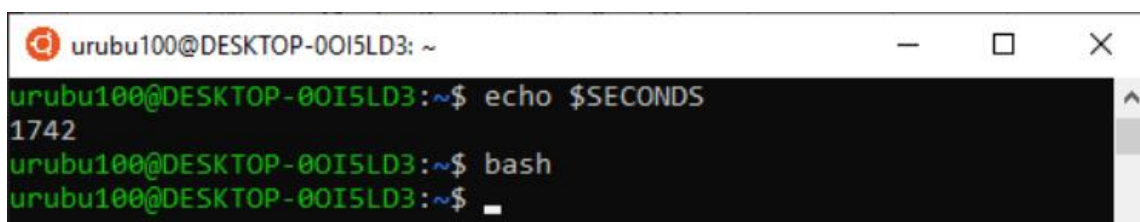
```
urubu100@DESKTOP-00I5LD3: ~  
urubu100@DESKTOP-00I5LD3:~$ echo $SECONDS  
1742  
urubu100@DESKTOP-00I5LD3:~$
```

Isto quer dizer que estamos com este bash ativo a 1742 segundos.

Vamos iniciar outra sessão de bash

Execute

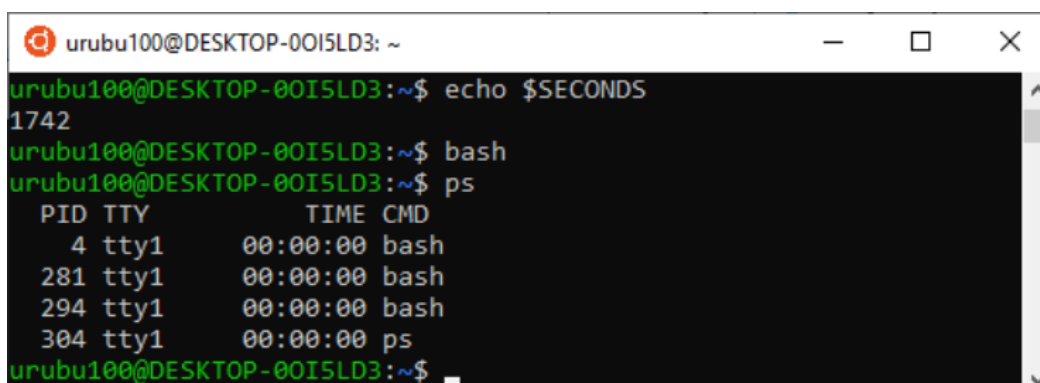
`bash`



```
urubu100@DESKTOP-00I5LD3: ~  
urubu100@DESKTOP-00I5LD3:~$ echo $SECONDS  
1742  
urubu100@DESKTOP-00I5LD3:~$ bash  
urubu100@DESKTOP-00I5LD3:~$
```

Você acredita que estamos com duas sessões de bashs ativas?

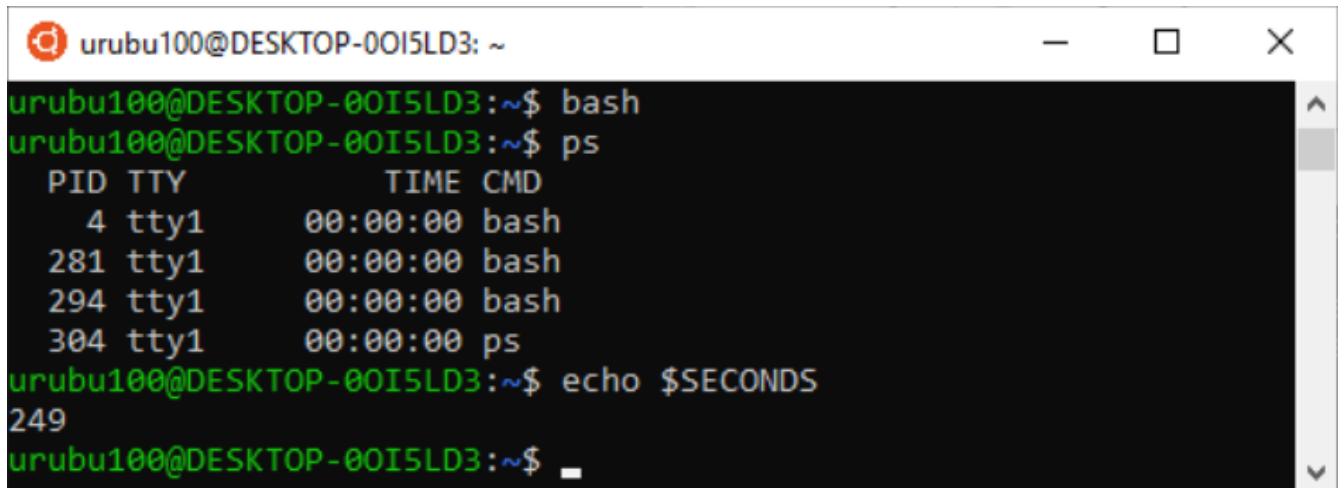
Sim, execute o comando `ps` (para listar os processos ativos)



```
urubu100@DESKTOP-00I5LD3: ~  
urubu100@DESKTOP-00I5LD3:~$ echo $SECONDS  
1742  
urubu100@DESKTOP-00I5LD3:~$ bash  
urubu100@DESKTOP-00I5LD3:~$ ps  
  PID TTY          TIME CMD  
    4 tty1      00:00:00 bash  
   281 tty1      00:00:00 bash  
   294 tty1      00:00:00 bash  
   304 tty1      00:00:00 ps  
urubu100@DESKTOP-00I5LD3:~$
```

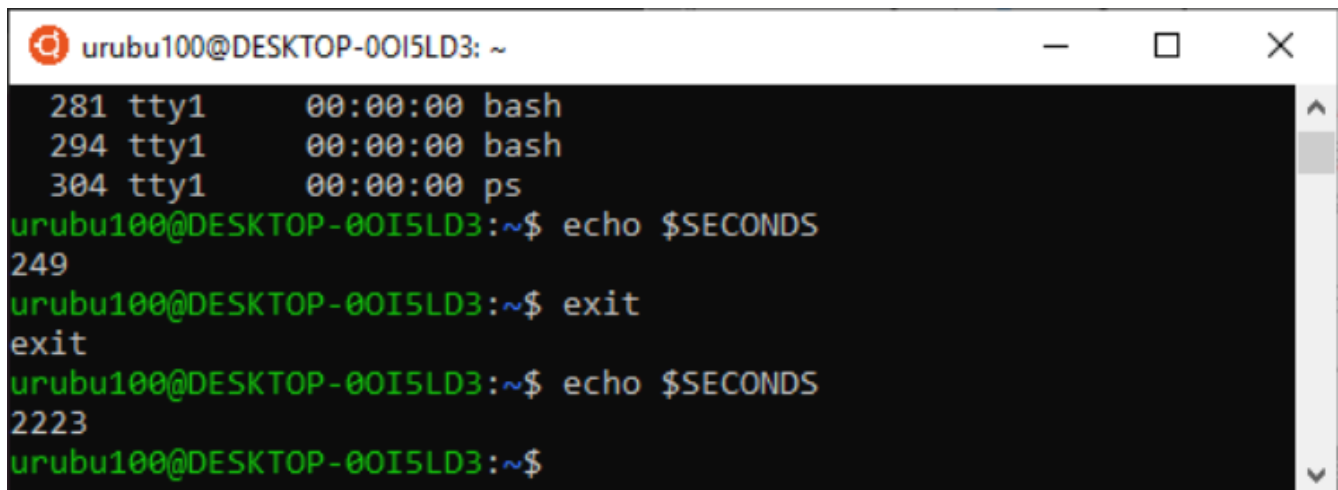
No meu caso, estou com três bashes diferentes, isto significa que podemos ter três variáveis de ambientes distintas, temos três bashes distintos, o que se fizer em um não será feito pelo outro, para isso precisamos exportar a variável de um bash a outro.

Se eu escrever a variável global SECONDS veja o que ocorre:



```
urubu100@DESKTOP-00I5LD3: ~  
urubu100@DESKTOP-00I5LD3:~$ bash  
urubu100@DESKTOP-00I5LD3:~$ ps  
  PID TTY          TIME CMD  
    4 tty1        00:00:00 bash  
   281 tty1        00:00:00 bash  
   294 tty1        00:00:00 bash  
   304 tty1        00:00:00 ps  
urubu100@DESKTOP-00I5LD3:~$ echo $SECONDS  
249  
urubu100@DESKTOP-00I5LD3:~$
```

O valor em segundos é menor porque o último bash foi ativado mais recentemente. Vamos sair com exit e usar a mesma variável local para o bash anterior, você verá que o tempo ficou maior, porque foi iniciado muito antes.



```
urubu100@DESKTOP-00I5LD3: ~  
  281 tty1        00:00:00 bash  
  294 tty1        00:00:00 bash  
  304 tty1        00:00:00 ps  
urubu100@DESKTOP-00I5LD3:~$ echo $SECONDS  
249  
urubu100@DESKTOP-00I5LD3:~$ exit  
exit  
urubu100@DESKTOP-00I5LD3:~$ echo $SECONDS  
2223  
urubu100@DESKTOP-00I5LD3:~$
```

Veja só o tempo de ativação como é muito maior, vamos sair deste bash

Se você executar **exit** novamente sairá do terminal.

A variável de ambiente **SHELL** apresenta o shell configurado atual como padrão.

```
urubu100@DESKTOP-00I5LD3: ~  
urubu100@DESKTOP-00I5LD3:~$ echo $SHELL  
/bin/bash  
urubu100@DESKTOP-00I5LD3:~$ _
```

Vamos criar variáveis locais e mostrar os subprocessos.

Execute:

script1='marise' #cria uma var local e atribui valor (não pode ter espaço)

echo \$script1 #mostra o conteúdo da variável criada (com \$ na frente veremos o conteúdo)

bash #cria um novo processo bash filho

ps #mostra os dois processos bash

echo \$script1 #mostra o conteúdo da variável criada

exit #volta para o processo pai

ps #mostra o processo único running

echo \$script1 #mostra o conteúdo da variável criada

export script1 #exporta a variável SCRIPT1

bash #cria um novo processa bash filho

ps #mostra os dois processos

echo \$script1 #mostra o conteúdo da variável global criada

Vamos ver essa execução no terminal:

```
urubu100@DESKTOP-00I5LD3:~$ echo $SHELL  
/bin/bash  
urubu100@DESKTOP-00I5LD3:~$ script1='marise'  
urubu100@DESKTOP-00I5LD3:~$ echo script1  
script1  
urubu100@DESKTOP-00I5LD3:~$ echo $script1  
marise  
urubu100@DESKTOP-00I5LD3:~$ _
```

```
urubu100@DESKTOP-00I5LD3:~$ bash  
urubu100@DESKTOP-00I5LD3:~$ ps  
  PID TTY          TIME CMD  
    4 tty1      00:00:00 bash  
   31 tty1      00:00:00 bash  
   41 tty1      00:00:00 ps  
urubu100@DESKTOP-00I5LD3:~$ _
```


Vamos procurar a variável script1 neste bash filho

```
urubu100@DESKTOP-00I5LD3:~$ echo $script1
urubu100@DESKTOP-00I5LD3:~$
```

Não está lá, retorna linha branco!

Porque a variável script1 é local e foi criada na bash pai

Vamos fechar o bashfilho

Execute o comando **exit**

Você volta para o bash pai, valide isso verificando se a variável script1, está nesse contexto.

```
urubu100@DESKTOP-00I5LD3:~$ exit
exit
urubu100@DESKTOP-00I5LD3:~$ echo $script1
marise
urubu100@DESKTOP-00I5LD3:~$
```

Exportar a variável local para que ela esteja disponível como global

```
urubu100@DESKTOP-00I5LD3:~$ export script1
urubu100@DESKTOP-00I5LD3:~$
```

Execute o comando **env**

```
urubu100@DESKTOP-00I5LD3:~$ env
SHELL=/bin/bash
TERM=xterm-256color
WSLENV=
USER=urubu100
NAME=DESKTOP-00I5LD3
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:
32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.l
;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;
=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31
=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35
ng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01
.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=0
gm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;3
ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=0
HOSTTYPE=x86_64
PATH=/home/urubu100/bin:/home/urubu100/.local/bin:/usr/local/sbin:
pps/CanonicalGroupLimited.Ubuntu16.04onWindows_2020.1604.14.0_x64_
hon27/Scripts:/mnt/c/Windows/System32:/mnt/c/Windows:/mnt/c/Window
ProgramData/chocolatey/bin:/mnt/c/Program Files/Git/cmd:/mnt/c/Pro
arise Miranda/AppData/Local/Programs/Microsoft VS Code/bin:/mnt/c/
PWD=/home/urubu100
LANG=en_US.UTF-8
SHLVL=1
HOME=/home/urubu100
script1=marise
LOGNAME=urubu100
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
LESSOPEN=| /usr/bin/lesspipe %s
LESSCLOSE=/usr/bin/lesspipe %s %s
_=/usr/bin/env
urubu100@DESKTOP-00I5LD3:~$
```

Neste ponto estamos com a variável exportada de forma global, qualquer bash filho herdará essa variável script1

Faça a sequência abaixo:

Verifique quantos bash estão ativos, no meu caso só o pai com **ps**

Execute **echo \$script1** para ver o conteúdo da variável

Crie um **bash** filho

Verifique se a variável de ambiente global foi herdada

Veja com o comando os que o pai tem o mesmo ID e há um novo filho bash, novo ID, que herdou a variável global.

Vamos remover a variável de ambiente script 1 no processo filho

Use o comando **unset script1**

Se você remove a variável do filho, não remove do pai

Termine esse processo filho com **exit** e crie outro que herde a variável global script1.

Depois remova a variável do pai e verifique se a variável no novo bash filho foi herdada.

```
urubu100@DESKTOP-00I5LD3:~$ ps
  PID TTY          TIME CMD
   4  tty1      00:00:00 bash
  43  tty1      00:00:00 ps
urubu100@DESKTOP-00I5LD3:~$ echo $script1
marise
urubu100@DESKTOP-00I5LD3:~$ bash
urubu100@DESKTOP-00I5LD3:~$ echo $script1
marise
urubu100@DESKTOP-00I5LD3:~$ ps
  PID TTY          TIME CMD
   4  tty1      00:00:00 bash
  44  tty1      00:00:00 bash
  54  tty1      00:00:00 ps
urubu100@DESKTOP-00I5LD3:~$ unset script1
urubu100@DESKTOP-00I5LD3:~$ echo $script1

urubu100@DESKTOP-00I5LD3:~$ exit
exit
urubu100@DESKTOP-00I5LD3:~$ ps
  PID TTY          TIME CMD
   4  tty1      00:00:00 bash
  55  tty1      00:00:00 ps
urubu100@DESKTOP-00I5LD3:~$ echo $script1
marise
urubu100@DESKTOP-00I5LD3:~$ _
```

Agora vamos remover a variável de ambiente global script1 e verificar se ela aparece no processo filho.

```
urubu100@DESKTOP-00I5LD3:~$ bash
urubu100@DESKTOP-00I5LD3:~$ echo $script1
marise
urubu100@DESKTOP-00I5LD3:~$ ps
  PID TTY          TIME CMD
   4 tty1        00:00:00 bash
  56 tty1        00:00:00 bash
  66 tty1        00:00:00 ps
urubu100@DESKTOP-00I5LD3:~$ exit
exit
urubu100@DESKTOP-00I5LD3:~$ ps
  PID TTY          TIME CMD
   4 tty1        00:00:00 bash
  67 tty1        00:00:00 ps
urubu100@DESKTOP-00I5LD3:~$ echo $script1
marise
urubu100@DESKTOP-00I5LD3:~$ unset script1
urubu100@DESKTOP-00I5LD3:~$ echo $script1

urubu100@DESKTOP-00I5LD3:~$ bash
urubu100@DESKTOP-00I5LD3:~$ ps
  PID TTY          TIME CMD
   4 tty1        00:00:00 bash
  68 tty1        00:00:00 bash
  78 tty1        00:00:00 ps
urubu100@DESKTOP-00I5LD3:~$ echo $script1

urubu100@DESKTOP-00I5LD3:~$ exit
exit
urubu100@DESKTOP-00I5LD3:~$ ps
  PID TTY          TIME CMD
   4 tty1        00:00:00 bash
  79 tty1        00:00:00 ps
urubu100@DESKTOP-00I5LD3:~$ echo $script1
```

Se você der um comando **env** verá que a variável global foi removida do sistema.

VARIÁVEIS DE AMBIENTE CONFIGURADA COMO ARRAY

script2=(azul amarelo verde vermelho) #cria o array

Para acessar a variável use **echo**, usamos as chaves e o índice de interesse e colchetes, observado que os índices iniciam pelo 0. Caso precise ver todos o conteúdo, tipo all, usamos ***** dentro dos colchetes.

```
urubu100@DESKTOP-00I5LD3:~$ script2=(azul amarelo verde vermelho)
urubu100@DESKTOP-00I5LD3:~$ echo $script2
azul
urubu100@DESKTOP-00I5LD3:~$ echo ${script2[2]}
verde
urubu100@DESKTOP-00I5LD3:~$ echo ${script2[*]}
azul amarelo verde vermelho
urubu100@DESKTOP-00I5LD3:~$
```


Para alterar o elemento ou acrescentar um valor ou remover

script2[3]=branco

```
urubu100@DESKTOP-00I5LD3:~$ script2[3]=branco
urubu100@DESKTOP-00I5LD3:~$ echo ${script2[*]}
azul amarelo verde branco
urubu100@DESKTOP-00I5LD3:~$
```

Remover o verde

unset script2[2]

```
urubu100@DESKTOP-00I5LD3:~$ unset script2[2]
urubu100@DESKTOP-00I5LD3:~$ echo ${script2[*]}
azul amarelo branco
urubu100@DESKTOP-00I5LD3:~$
```

FUNÇÕES #execute-as como root

nome_função(){comandos}

script4(){

>cd /home/marise

>echo 'incluir linha'>> func1

>}

```
root@DESKTOP-00I5LD3:~# script4(){
> cd /home/marise;
> echo 'incluir linha' >> func1
> }
root@DESKTOP-00I5LD3:~# typeset -f
```

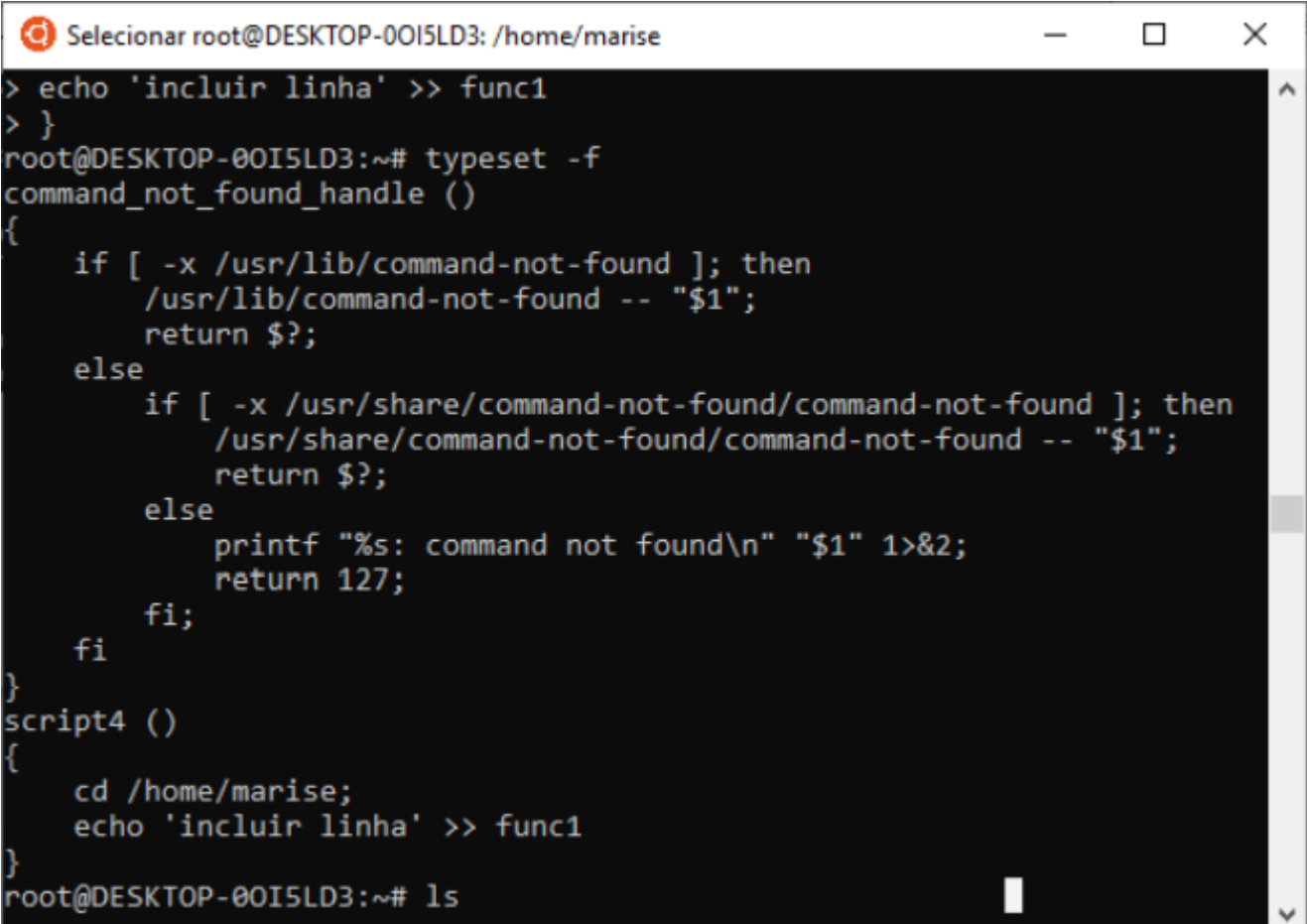
Detalhes

> é a forma como escrevemos o script da função

>> anexar um arquivo

Vamos verificar se a função foi criada:

typeset -f



```
Selecionar root@DESKTOP-00I5LD3: /home/marise
> echo 'incluir linha' >> func1
> }
root@DESKTOP-00I5LD3:~# typeset -f
command_not_found_handle ()
{
    if [ -x /usr/lib/command-not-found ]; then
        /usr/lib/command-not-found -- "$1";
        return $?;
    else
        if [ -x /usr/share/command-not-found/command-not-found ]; then
            /usr/share/command-not-found/command-not-found -- "$1";
            return $?;
        else
            printf "%s: command not found\n" "$1" 1>&2;
            return 127;
        fi;
    fi;
}
script4 ()
{
    cd /home/marise;
    echo 'incluir linha' >> func1
}
root@DESKTOP-00I5LD3:~# ls
```

Procure no final do arquivo, esta listagem é muito longa, veja se a função foi criada.

Agora chame a função **script4**

```
root@DESKTOP-00I5LD3:~# script4
root@DESKTOP-00I5LD3:/home/marise# script4
root@DESKTOP-00I5LD3:/home/marise# ls
forksample func1
root@DESKTOP-00I5LD3:/home/marise# func1
func1: command not found
root@DESKTOP-00I5LD3:/home/marise# ls
forksample func1
root@DESKTOP-00I5LD3:/home/marise# cat func1
incluir linha
incluir linha
root@DESKTOP-00I5LD3:/home/marise# cat func1
incluir linha
incluir linha
root@DESKTOP-00I5LD3:/home/marise# cat func1
incluir linha
incluir linha
root@DESKTOP-00I5LD3:/home/marise#
```

Se você executar **ls -a** #all mostras todos os ocultos que estão com pontinho na frente

Os bashes aqui são arquivos de configuração

Entre no seu usuário e execute o comando **ls -a**

```
urubu100@DESKTOP-00I5LD3:~$ ls -a
.          DESKTOP-00I5LD3_200415_2203.nmon
..         func1
.bash_history .local
.bash_logout .nano
.bashrc      .profile
.bashrc.save script1.sh
..bashrc.swp .sudo_as_admin_successful
.cache       .viminfo
.config      .wget-hsts
```

- **Azul:** Diretório
- **Verde:** arquivo de dados executável ou reconhecido
- **Céu Azul:** arquivo de link simbólico
- **Amarelo com fundo preto:** dispositivo
- **Rosa:** arquivo de imagem gráfica
- **Vermelho:** arquivo morto
- **Vermelho com fundo preto:** link quebrado
- Arquivos ocultos ponto na frente “.”

Vamos mostrar o conteúdo do .bashrc

vi .bashrc

Vá com a seta prompt até encontrar os alias incluídos nos scripts

Os comandos deste arquivo são executados no momento que o usuário inicia um shell com as características acima

```
alias grep='grep --color=auto'
alias fgrep='fgrep --color=auto'
alias egrep='egrep --color=auto'
fi

# colored GCC warnings and errors
#export GCC_COLORS='error=01;31:warning=01;35:note=01;36:caret=01;32:loc
us=01:quote=01'

# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
# sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal
|| echo error)" "$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[\;
&
]\s*\$*alert$//'\`)'"'

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
. ~/.bash_aliases
fi
```

105,1 88%

Vamos executar o alias **ll** onde está marcado com o quadrinho branco na imagem anterior

```
[8]+ Stopped vi .bashrc
urubu100@DESKTOP-00I5LD3:~$ ll
total 228
drwxr-xr-x 1 urubu100 urubu100 4096 Jun  2 16:35 ./
drwxr-xr-x 1 root root 4096 May 20 20:17 ../
-rw-r--r-- 1 urubu100 urubu100 3474 Jun  2 15:26 .bash_history
-rw-r--r-- 1 urubu100 urubu100 16384 Jun  2 16:34 .bash_history.swp
-rw-r--r-- 1 urubu100 urubu100 220 Mar 17 13:48 .bash_logout
-rw-r--r-- 1 urubu100 urubu100 12288 Jun  2 16:32 .bash_logout.swp
-rw-r--r-- 1 urubu100 urubu100 3771 Mar 17 13:48 .bashrc
-rw-r--r-- 1 urubu100 urubu100 3792 Apr 23 01:01 .bashrc.save
-rw-r--r-- 1 urubu100 urubu100 16384 Jun  2 16:35 .bashrc.save.swp
-rw-r--r-- 1 urubu100 urubu100 16384 Jun  2 16:35 .bashrc.swo
-rw-rw-rw- 1 urubu100 urubu100 1024 Apr 23 01:00 .bashrc.swp
-rw-r--r-- 1 urubu100 urubu100 16384 Jun  2 16:31 .bashrc.swp
drwx----- 1 urubu100 urubu100 4096 Jun  2 16:33 .cache/
drwx----- 1 urubu100 urubu100 4096 Jun  2 16:33 .config/
-rw-rw-rw- 1 urubu100 urubu100 122814 Apr 15 23:02 DESKTOP-00I5LD3_20041
5_2203.nmon
-rw-rw-rw- 1 urubu100 urubu100 28 Jun  2 16:10 func1
drwx----- 1 urubu100 urubu100 4096 Mar 31 15:36 .local/
drwxrwxrwx 1 urubu100 urubu100 4096 Apr 23 00:59 .nano/
-rw-r--r-- 1 urubu100 urubu100 655 Mar 17 13:48 .profile
-rw-r--r-- 1 urubu100 urubu100 4096 Jun  2 16:33 .profile.swp
-rwxrwxrwx 1 urubu100 urubu100 37 May 18 23:47 script1.sh*
-rw-r--r-- 1 urubu100 urubu100 0 Apr 14 00:29 .sudo_as_admin_succes
sful
drwxr-xr-x 1 urubu100 urubu100 4096 Jun  2 16:33 .vim/
-rw-r--r-- 1 urubu100 urubu100 684 May 21 12:35 .viminfo
-rw-rw-rw- 1 urubu100 urubu100 167 Apr  1 01:24 .wget-hsts
```