Resumo Importante sobre o Curso de LPIC-1

Tópico 103.1 - Trabalhando na Linha de Comando – Variáveis de Ambiente

Douglas Luna - @DipionX02

O que estudamos nesta aula?

Introdução sobre variáveis de ambiente

Variável de ambiente é uma variável de um sistema operacional que geralmente contém informações sobre o sistema, caminhos de diretórios específicos no sistema de arquivos e as preferências do utilizador. Ela pode afetar a forma como um processo se comporta, e cada processo pode ler e escrever variáveis de ambiente.

Em todos os sistemas Unix, cada processo possui seu conjunto privado de variáveis de ambiente. **Por padrão, quando um processo é criado ele herda uma cópia das variáveis que foram exportadas no ambiente do processo pai.** Todos os tipos de Unix assim como o DOS e o Microsoft Windows possuem variáveis de ambiente; entretanto, variáveis para funções parecidas entre os sistemas possuem nomes distintos. Programas podem acessar os valores das variáveis de ambiente para efeitos de configuração.
Shell scripts e arquivos de lote usam variáveis para armazenar dados temporários e também para comunicar dados e preferências a processos filhos.

No Unix, as variáveis de ambiente são normalmente inicializadas durante a inicialização do sistema, e no início de cada sessão. Esse assunto será estudado melhor no tópico 5.

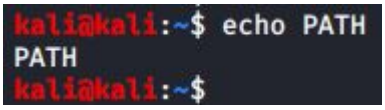
As variáveis podem ser usadas tanto por scripts quanto pela linha de comando. São geralmente referenciadas usando-se símbolos especiais na frente ou nas extremidades no nome da variável. Por exemplo, **Unix usa-se o $.** O **cifrão**, quando iniciamos o comando identifica que é uma variável de ambiente, se não colocarmos o cifrão ele apenas imprime na tela o conteúdo, como vimos na aula anterior:

Se dermos o comando abaixo, ele nos retornará o caminho de cara path de variável no sistema:

```
$ echo $PATH
```

Sem o cifrão ele apenas imprime a palavra na tela:

```
$ echo PATH
```

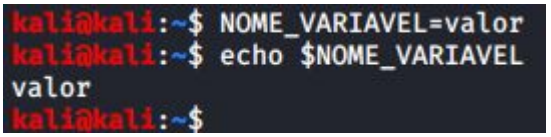


Como declarar uma variável

Para declarar (criar) uma variável no ambiente Linux seguimos o princípio de qualquer linguagem de programação, **primeiros damos o nome e depois o valor**. Exemplo no terminal:

```
$ NOME_VARIAVEL=valor
```

Acima dizemos que o nome da variável é **NOME_VARIAVEL** e depois declaramos o seu valor, ou seja, NOME_VARIAVEL é o mesmo que ***valor.***

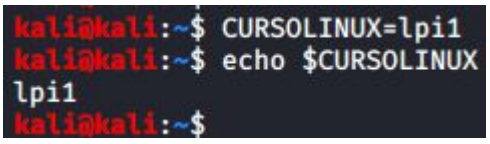


Para ficar mais claro, vamos declarar mais uma variável:

```
$ CURSOLINUX=lpi1
```

Se digitarmos no terminal o comando abaixo ele nos voltar o valor de atribuímos a variável **CURSOLINUX** que é *lpi1*:

```
$ echo $CURSOLINUX
```



Note que, primeiro declaramos a variável e logo depois consultamos qual o valor dela.

Variáveis Locais e Globais

Importante frisar que, quando declaramos uma variável da forma que é explicado acima, ele fica disponível e visível somente em âmbito local, ou seja, a variável só é visível na sessão do shell que está aberta, quaisquer seções abertas posteriormente não encontrarão as variáveis criadas.

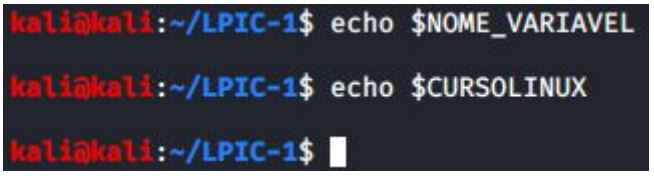
Vejamos, como já criamos as variáveis **NOME_VARIAVEL** e **CURSOLINUX** vamos iniciar um novo bash dentro do próprio bash que está aberto, lembrando que estas variáveis ainda estão em nível local:

```
$ bash
```

Note que estamos em outra seção, agora faça o teste e tente consultar o valor das variáveis NOME_VARIAVEL e CURSOLINUX:

```
$ echo $NOME_VARIAVEL
```

```
$ echo $CURSOLINUX
```



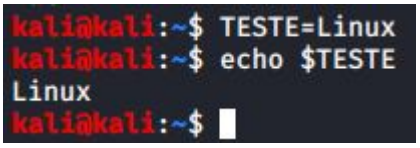
Não nos voltará nenhum valor, pois estas variáveis estão armazenadas apenas localmente para aquele usuário e aquela seção anterior. Para voltar ao **bash** anterior digite “**exit**”. Isso vale também caso o **bash** seja fechado, ou seja, se fecharmos o terminal as variáveis também serão apagadas.

O comando export

Para que os processos filhos consigam ver essas variáveis (globais) usamos o comando **export** (export NOME_VARIAVEL), pois só após a exportação dessas variáveis é que os processos filhos poderão enxergar as mesmas, seguindo a hierarquia. Toda seção que for aberta durante o bash atual, será “filha” dela, assim herdando as variáveis**. O comando export só funciona para processos que são originados a partir do bash atual.** Cuidado para não confundir com um novo terminal, pois é totalmente independente.

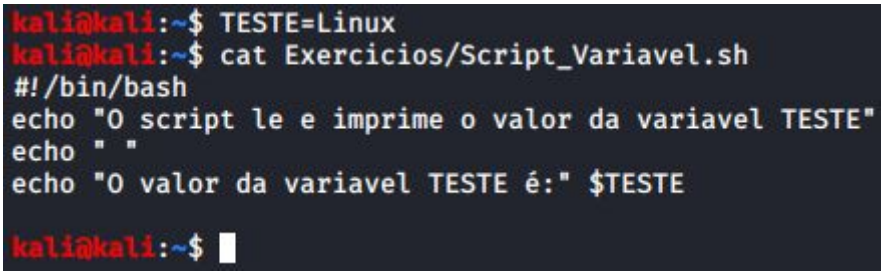
Vamos declarar a variável TESTE=Linux:

$ TESTE=Linux



Agora, vamos usar o arquivo da aula para testar a variável, o arquivo se chama Scritp_Variavel.sh, vamos ver o que tem dentro dele:

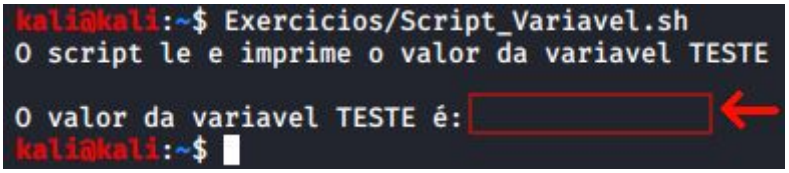
$ cat Script_Variavel.sh



Podemos ver que durante o script há campo chamando o valor da variável TESTE, vamos lembrar que quando tem o **cifrão “$”** quer dizer que é uma variável.

Se fizermos o comando, vamos ver que ficará um campo sem o valor da variável.

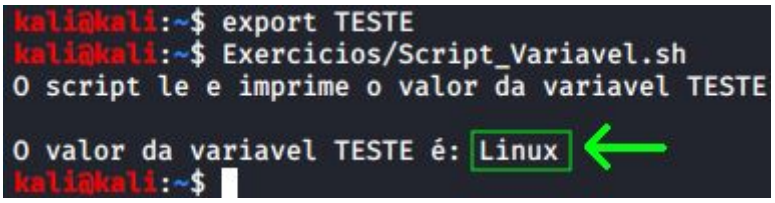
$ ./Script_Variavel.sh



Isso acontece porque a variável está local, então somente o bash atual consegue encontrá-lo. Para que toda seção a partir da atual passe a enxergar a variável **TESTE** temos que fazer um **export**.

```
$ export TESTE
```

Agora, se rodarmos novamente o script do exercício temos o seguinte resultado:



Podemos criar a variável e depois fazer o export dela, ou podemos criar e exportar diretamente uma variável com o comando abaixo.

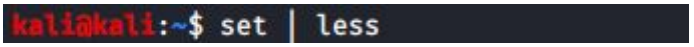
```
$ export NOME_VARIAVEL=valor
```

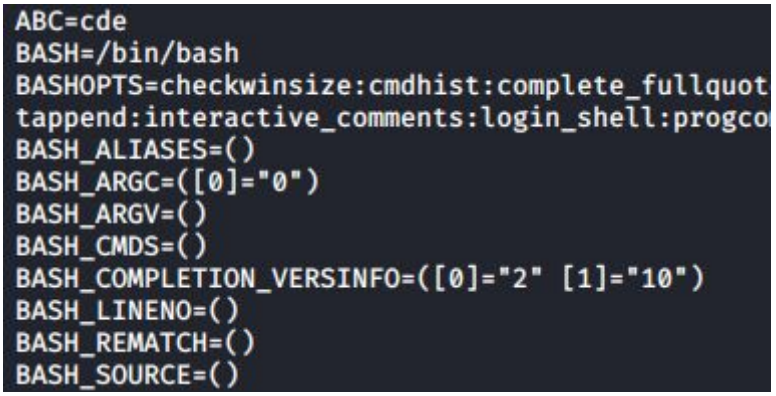
Verificar as variáveis

O comando set

O comando **set** irá mostrar todas as variáveis, as locais e as globais declaradas no bash atual. Ou seja, todas que foram iniciadas automaticamente, que foram declaradas manualmente, e que são ou não exportadas.

$ set | less





Ainda não estudamos o comando parâmetro **less**, virá nas próximas aulas. Mas ele é usado para minimizar a quantidade de informações na tela. Após o comando, veremos várias variáveis de ambiente inclusive que que declaramos.

O comando env

O comando **env mostrará apenas as variáveis globais.**

Diferenças os comandos entre set e env

A grande diferença entra o set e o **env** é que o **set** é um comando de origem do **bash**, já o **env** é um comando de uma aplicação externa, por isso ele enxerga somente as globais.

```
kali@kali:~$ type env
env is /usr/bin/env
kali@kali:~$ type set
set is a shell builtin
kali@kali:~$
```

Vamos declarar uma variável ABC com o valor cde (ABC=cde).

```
kali@kali:~$ ABC=cde
kali@kali:~$
```

Agora, vamos dar o comando para visualizar as variáveis locais e globais, o **set**.

```
$ set | less
```

De cara, já conseguimos ver a variável que declaramos:

```
ABC=cde
BASH=/bin/bash
BASHOPTS=checkwinsize:
tappend:interactive_co
BASH_ALIASES=()
```

Agora, vamos fazer a mesma coisa, porém com o comando **env**:

```
$ env | less
```

```
SHELL=/bin/bash
SESSION_MANAGER=local/kali:@/tmp/.ICE-u
WINDOWID=0
QT_ACCESSIBILITY=1
XDG_CONFIG_DIRS=/etc/xdg
XDG_SESSION_PATH=/org/freedesktop/Displ
XDG_MENU_PREFIX=xfce-
TMUX=/tmp//tmux-1000/default,21315,0
LANGUAGE=
```

Note que já não conseguimos ver a variável **ABC**, pois o comando **env** mostra somente as variáveis que são globais. Para que o comando **env** a encontre temos que fazer o **export,** veja:

$ export ABC=cde

```
kali@kali:~$ export ABC
kali@kali:~$ export ABC=cde
```

Agora vamos repetir o comando **env**:

$ env | less

```
kali@kali:~$ env | tail
XDG_RUNTIME_DIR=/run/user/1000
XDG_DATA_DIRS=/usr/share/xfce4:/usr/l
PATH=/usr/local/sbin:/usr/sbin:/sbin:
TESTE=Linux
GDMSESSION=lightdm-xsession
DBUS_SESSION_BUS_ADDRESS=unix:path=/r
_JAVA_OPTIONS=-Dawt.useSystemAAFontSe
ABC=cde
OLDPWD=/home/kali/LPIC-1
 =/usr/bin/env
```

Isso acontece porque agora a variável está disponível globalmente.

Uma outra opção do **env** é que, ele pode alterar o valor uma variável de forma temporária, por exemplo, com o script do exercício, o valor da variável TESTE é Linux, vamos alterá-lo apenas na execução. (*Lembrando que estamos dentro do diretório Exercícios*):

$ env TESTE=Windows ./Script_Variavel

```
kali@kali:~/Exercicios$ env TESTE=Windows ./Script_Variavel.sh
O script le e imprime o valor da variavel TESTE

O valor da variavel TESTE é: Windows
kali@kali:~/Exercicios$
```

Agora se executarmos novamente o comando de forma normal, ele apresentará o valor verdadeiro da variável.

$ ./Script_Variavel

```
kali@kali:~/Exercicios$ ./Script_Variavel.sh
O script le e imprime o valor da variavel TESTE

O valor da variavel TESTE é: Linux
kali@kali:~/Exercicios$
```

Como remover variáveis criadas

Da mesma forma que temos como criar uma variável, também temos como removê-las do sistema. Para isso usamos o comando **“unset”**.

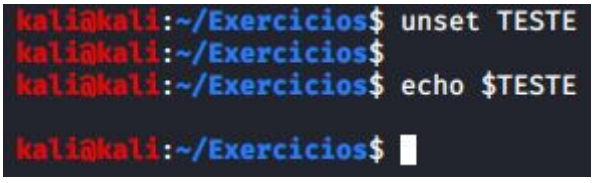
O comando unset

Sempre que quisermos excluir uma variável usamos o comando **“unset”**. Para remover o valor da variável TESTE usaremos o comando:

```
$ unset TESTE
```

Agora, vamos consultar a variável com o comando echo e veremos que não teremos mais o valor da variável TESTES disponível.

```
$ echo $TESTE
```



Variáveis Pré-definidas do ambiente Linux

Ainda sobre variáveis vale frisar que existem variáveis que já são predefinidas no sistema Linux, as quais, são carregadas durante o sistema, na aula citamos algumas que inclusive podem ser cobradas no exame.

```
$ set | more
```

Aqui vão as Principais Variáveis de Ambiente:

- **DISPLAY**: Indica às aplicações gráficas onde as janelas deverão ser exibidas. Será estudado no Tópico 106.
- **HISTFILE**: Arquivo do histórico de comandos
- **HISTFILESIZE**: Quantidade de linhas/comandos armazenados no arquivo de histórico
- **HOME**: indica o diretório do usuário atual
- **LOGNAME e USER**: Nome do usuário atual
- **PATH:** Diretórios em que o Linux irá procurar por arquivos executáveis
- **PS1:** Aparência do prompt do shell.
- **PWD**: Diretório atual
- **OLDPWD:** Diretório anterior
- **SHELL:** Qual shell está sendo utilizado
- **TERM**: xterm - Mostra qual terminal estamos usando, no caso estamos usando interface gráfica. Caso optemos por logar sem passar por interface gráfica, aparecerá **TERM**=tty.

Todas as variáveis podemos conferir com o comando echo.

Variáveis dinâmicas

Existem algumas variáveis de ambiente que são definidas dinamicamente pelo SHELL, é importante que conheçamos elas. Elas são identificadas pelo cifrão no início, por exemplo:

Este comando mostra o PID do processo atual.

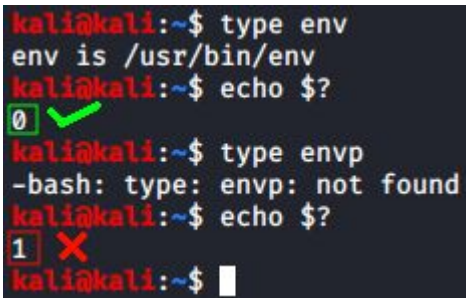
```
$ echo $$
```

Este comando mostra o PID do último processo que executamos em background.

```
$ echo $!
```

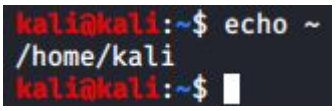
Este comando mostra o código de saída (exit code) do último do processo executado. Temos o retorno de 0, quando o último comando foi executado com sucesso e 1 ou 2 com foi com falha (qualquer número diferente de 0 significa falha).

$ echo $?



Ainda temos o “~”, que contém o /home do usuário, seja ele qual for.

$ echo ~



Ainda podemos apontar outro usuário com o mesmo comando, e assim obtemos o home deste usuário:

$ echo ~root

Quando executamos o comando **cd ~,** vamos direto para o /home do usuário atual.
Material adicional:

$ cd ~