06

Lidando com váriaveis de shell e de ambiente

Shell and environment variables with export

O próximo tópico que iremos abordar são as variáveis do *shell* e da linha de comando no geral.

Vamos ao terminal, iremos digitar `echo Guilherme, bem vindo` . Ao dar um "enter" ele nos mostra a mensagem reproduzida da mesma maneira: `echo Guilherme, bem vindo!` . Veremos em nossa tela o seguinte:

```
echo Guilherme, bem vindo!
Guilherme, bem vindo!
```

E se quisermos utilizar nesse comando algum tipo de variável?

O *shell* executa comandos e permite diversas características de uma linguagem de programação. Uma delas é o uso de variáveis.

Como criamos uma variável no *shell*?

Para criar uma variável no *shell* basta dizer: "nome da variável=o valor da variável". Por exemplo, vamos supor que eu tenha 34 anos, então, idade=34, temos o nome da variável "idade" e seu respectivo valor "34". Se dermos um enter ele vai criar a variável idade.

```
echo Guilherme, bem vindo!
Guilherme, bem vindo!
idade=34
```

E agora, como utilizamos a variável idade?Vamos digitar `echo Guilherme tem $idade anos` e se dermos um "Enter" na próxima linha teremos `Guilherme tem 34 anos`

```
echo Guilherme, bem vindo!
Guilherme, bem vindo!
idade=34
echo Guilherme tem $idade anos
Guilherme tem 34 anos
```

Observe o cifrão na frente de "idade". Ele serve para dizer que a partir do ponto em que ele está existe uma variável que deve ser substituída pelo seu valor real. Assim, por exemplo, se quiséssemos acrescentar mais uma variável, uma variável "nome", poderíamos digitar: echo $nome tem $idade anos.

```
echo $nome tem $idade anos
tem 34 anos
```

Observe que `$nome` foi interpretado pelo *shell* como algo que não possui um valor, que é vazio. Bom, isso ocorre, pois, ainda não declaramos que essa variável possui algum valor. Por esse motivo, tivemos, apenas, o seguinte retorno: "tem 34 anos". O

que aconteceu é que apenas o valor da variável idade foi lido, uma vez que já atribuímos um valor a ela. Agora, podemos adicionar um valor a variável *nome*, ele será `nome=Guilherme`.

Podemos testar nossa nova variável, `echo $nome tem $idade anos`:

```
nome=Guilherme
echo $nome tem $idade anos
Guilherme tem 34 anos
```

Como podemos observar ele trocou o `$` pelo valor da variável que desejamos, tanto no nome quanto na idade.

Vamos alterar o valor da variável `nome` e ver o que ocorre. Digitaremos `nome= Paulo`. Se executarmos essa variável dando um "Enter", teríamos a resposta de que esse comando não foi encontrado. E se alterarmos o valor de `idade`? Vamos digitar que `idade= 35`. Teremos a mesma resposta, `35: command not found`.

```
nome= Paulo
Paulo: command not found
idade= 35
35: command not found
```

Por que ele não encontrou esses comando?

Muito cuidado ao declarar uma variável no *shell*.

Repare que a primeira vez que declaramos a variável, tínhamos digitado o seguinte: `idade=34` e agora digitamos, `idade= 35`. Note que existe uma pequena diferença de um para o outro, o fato de existir um espaço entre o `=` e `35`. Ao acrescentarmos um espaço antes ou depois do `=` estamos passando a indicação de que ele deve, na verdade, executar o comando `35` ou o comando `Paulo`. Por isso, ele nos responde que esses dois comandos não existem.

E se o espaço ficasse antes do `=`? Se digitássemos, por exemplo, `nome =Paulo`, ele nos responderia o seguinte:

```
nome =Paulo
No command `nome` found, did you mean:
    Command `node` from package `nodejs-legacy` (universe)
    Command `node` from package `node` (universe)
    Command `tome` from package `tome` (multiverse)
    Command `note` from package `note` (universe)
nome: command not found
```

Mais uma vez ele ficará confuso. Agora ele está buscando executar o comando `nome`, é como se fossemos digitar o `ls` e mais alguma coisa, ou o `echo` e um complemento e etc. A primeira palavra na nossa ordem é sempre um comando. Lembre-se que a ordem da sintaxe de um comando no *shell* é, primeiro, o nome do comando, segundo, as opções e, por fim, os argumentos. As últimas duas nem sempre nessa ordem, mas em geral é esse o padrão que podemos encontrar. Por isso, ao digitarmos um espaço antes do `=`, como em `idade =34` ele interpreta `idade` como o argumento. E se digitarmos o espaço depois do `=`, como `idade= 34`. Ele tentará executar o comando `34`. E se fizermos simplesmente o `idade=34`, finalmente, teremos o valor `34` para idade.

Repare que mesmo limpando a tela algumas vezes, podemos usar `echo $nome tem $idade anos` e teremos como resposta:

```
echo $nome tem $idade anos
Guilherme tem 34 anos
```

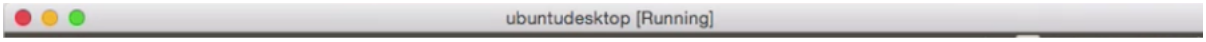
Limpar a tela não está relacionado com limpar as variáveis. Elas continuarão ocorrendo, mesmo depois de termos dado `clear` na nossa tela algumas vezes.

O `$nome` e a `$idade` são variáveis do *shell*, isto é, elas existem nesse *shell* atual. Se rodarmos um programa dentro do *shell*, esse programa não verá essas variáveis. Isso ocorre porque ele é um outro *shell*, ou seja, é um outro programa. Este outro programa executado não terá acesso as variáveis do *shell*, por exemplo, as variáveis `nome` e `idade`. Temos acesso a essas variáveis apenas dentro desse nosso *shell*. E se quiséssemos buscar todos os arquivos em um formato específico, por exemplo, se quero executar todas as listagens com `ls -la`, colocando isso dentro de uma variável, para que, a formatação venha em forma de lista? Podemos criar uma variável `padrao=-la`. Dando um "Enter" teremos criado essa variável.

Para confirmar que ela existe podemos digitar `echo $padrao`

```
padrao=-la
echo $padrao
-la
```

Se fizermos ainda um `ls $padrao`. Ele estará executando um `ls -la` e teremos a listagem em formato de tabela:



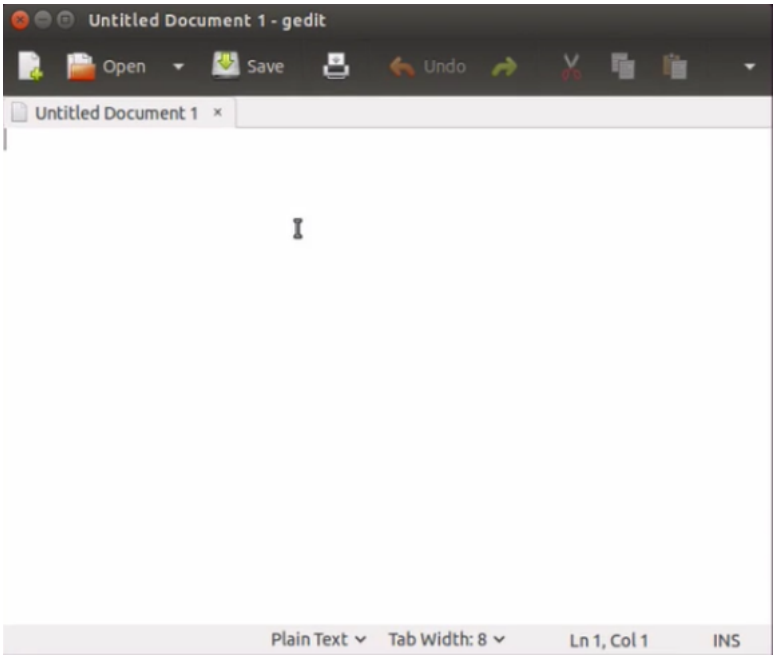
Podemos utilizar a vontade no *bash*, as variáveis de *bash*. Vamos observar o que acontece quando digitamos `ls $padrao`. Agora, que temos que tomar cuidado. O shell vai executar o comando `ls $padrao`, isso significa que o `shell` interpreta cada um dos argumentos, `ls` é o `ls`, o `$padrao` é `-la`. Então, ele entende através do `ls $padrao` para executar "ls - la". Dentro da execução do `ls`, nós não temos acesso a variável `$padrao`, pois está é uma variável do *shell*. Só quem tem acesso a isso é

o próprio *shell* antes de executar o `ls` . É o *shell* quem interpreta os cifrões das variáveis, isto é, seus nomes, ele é quem colocará as variáveis nos lugares adequados e é ele também quem executa o `ls -la` . Um programa externo não tem acesso as variáveis que foram definidas no *shell*.

Mas, como provamos que se executarmos um programa aqui dentro ele não tem acesso as variáveis?

Uma maneira de provar isso é criando um script, um programa, onde iremos tentar utilizar essa variável. Vamos utilizar um editor de texto, o `gedit` .

Após termos ele aberto, iremos gravar um arquivo.



Iremos digitar no editor `echo $nome tem $idade anos` . Vamos salvar esse arquivo no "Save As" com o nome de "mostra_idade". Falaremos sobre `scripts` mais adiante, em uma seção onde trataremos sobre esse tema. O que queremos demostrar é que quando tentarmos executar no *shell* esse script que acabamos de criar, não será possível, pois ele não tem acesso as variáveis `$nome` e `$idade` .

Voltando para o terminal , ao tentar executar o `script` chamando `ls` teremos o seguinte:

```
ls
Desktop     Dowloads             mostra_idade         Pictures       Templates
Documents   examples.desktop     Music                Public         Videos
```

Observe que na listagem aparece nosso script: `mostra_idade`. Uma maneira de executar um script é pedir ao *bash* para fazer isso. Digitaremos `bash mostra_idade`:

```
ls
Desktop     Dowloads             mostra_idade         Pictures       Templates
Documents   examples.desktop     Music                Public         Videos
bash mostra_idade
tem anos
```

Mas, onde está o nome e a idade? Sumiram! Pois, as variáveis quando são criadas, por padrão são variáveis do *shell*, ou seja, só vivem nesse *shell*. Quando executamos um processo dentro desse *shell*, quando os comandos são interpretados, tudo bem, temos acesso as variáveis pois é o *shell* que faz isso. Mas, no momento em que executamos o comando, durante a execução do comando, não é possível termos acesso as variáveis do shell, do processo que as invocou.

É extremamente importante lembrarmos disso!

É claro que, se é isso que acontece, é preciso de um recurso que permita que peguemos as variáveis de *shell* e disponibilizemos elas para quem for sendo executado dentro do *shell*. Então, vamos disponibilizar essas duas variáveis para o "mostra idade", isto é, para quem está sendo executado. Se quisermos fazer isso podemos falar para o *shell* para ele exportar a variável que quisermos, por exemplo, `export nome`. Faremos o mesmo com as outras variáveis, no caso, `export idade` e, por fim, rodaremos `bash mostra idade` e ele irá nos devolver, finalmente, `Guilherme tem 34 anos`

```
export nome
export idade
bash mostra idade
Guilherme tem 34 anos
```

Retomando: O que foi que fizemos?

Começamos com uma variável de *shell*, lembrando que é extremamente importante tomar cuidado com o espaço quando digitamos as variáveis, isto é, não deixando espaços antes ou depois do `=` para que o *shell* não interprete o que vêm antes ou depois como argumento. Em seguida, o que fizemos foi exportar a variável. É importante exportar as variáveis, caso contrário, os processos filhos deste *shell* não tem acesso as variáveis de *shell*. Se exportamos a variável `nome`, através do `export nome` chamaremos essa variável de "ambiente", ela possui esse valor. Agora sim, quando chamarmos os processos filhos o `bash mostra_idade`, terá acesso as variáveis de "ambiente", que tem o valor `Guilherme` e `34`. É importante diferenciarmos as variáveis de *shell* das variáveis de ambiente.

Como fazemos para pegar uma variável de *shell* e exportar ela como variável de "ambiente"?

Podemos digitar `export`, damos um espaço e digitamos o nome da variável.

Veremos, mais adiante, diversas maneiras de trabalhar com as variáveis de *shell* e de *ambiente*.

Mais do export e do env

Vimos que podemos criar variáveis de ambiente, por exemplo, as variáveis `$nome` e `$idade` que valem, respectivamente, "Guilherme" e "34". E podemos exportar essas variáveis usando o `export`, digitando `export nome` e `export idade` e executar o `bash` e pedir para ele executar o comando `mostra_idade` e teremos `Guilherme tem 34 anos`

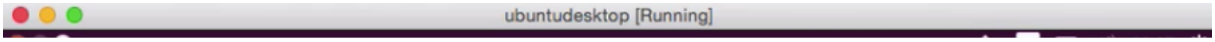
```
echo $nome
Guilherme
echo $idade
34
export nome
export idade
bash mostra_idade
```

Por que chamamos o comando de `bash`?

Porque ele é que estamos rodando como *shell* no *Ubuntu*, o comando `bash` existe aqui dentro e estamos pedindo para que se execute o `bash` como um processo filho do `script` `mostra_idade`. Como a variável em questão é uma variável de ambiente ele consegue achar.

Como fazemos para zerar o que já fizemos?

Basta fechar nosso atual terminal e abrir um novo:



No terminal novo repare que estamos executando um *shell* totalmente novo que não tem relação com o *shell* que estavamos utilizando até esse momento. Nesse novo terminal, portanto, não teremos as variáveis `$nome` nem `$idade`. Se tentarmos executar o `mostra_idade`, entretanto, ele irá nos mostrar só `tem anos` e não mais nada.

```
echo $nome
```

```
echo $idade

bash mostra_idade
tem anos
```

Vamos criar uma variável nova, `nome=Guilherme` , e executar isso, digitando `bash mostra_idade` . Teremos:

```
nome=Guilherme
bash mostra_idade
tem anos
```

E ele continuará sem falar nada. Por que isso acontece? Isso ocorre porque a variável é de *shell* e não de ambiente. Para transformar essa variável em uma de ambiente temos que exportar, para isso, digitamos `export nome` . Vamos, mais uma vez, escrever o `bash mostra_idade` e ver o que acontece:

```
nome=Guilherme
bash mostra_idade
tem anos
export nome
bash mostra_idade
Guilherme tem anos
```

A variável de idade ainda não aparece, pois, não criamos ainda ela, mas o nome, como podemos perceber, já está constando.

Toda vez que quisermos criar uma variável de `ambiente` o padrão é esse?

É preciso digitar "nome=alguma coisa" e "export alguma coisa". Podemos utilizar um atalho, digitando `export nome=Paulo` . Com isso já estamos falando que a variável que estamos criando, que é de *enviroment* (ambiente), já vai se chamar nome e ter o valor Paulo. Vamos chamar o `bah mostra_idade` e ver o que acontece:

```
export nome=Paulo
bash mostra_idade
Paulo tem anos.
```

E para a variável de *shell*? Ela também será "Paulo"? Vamos fazer o teste:

```
echo Paulo
Paulo
```

Vemos que a variável de *shell* também é "Paulo".

Repare que o que acontece é que temos uma variável chamada "nome" do *shell* e ela possui o valor "Guilherme". Quando utilizamos `export` o que dizemos para ela é que é uma variável de ambiente que tanto pode atuar no *shell* atual quanto nos processos filhos desse *shell*. Se falamos direto `export nome=Paulo` estamos falando que a variável "nome" vai ter o valor local (de *shell*) e de ambiente, ela valerá "Paulo". Por isso, quando executado, ele mostra o valor "Paulo". Quando tentamos mostrar só a variável no nosso próprio *shell* ele já mostra Paulo também.

Bom, agora, já temos as variáveis. Vimos duas maneiras de exportar: (1) utilizando apenas o `export` e (2) podemos exportar através do "export + nome da variável=valor".

Mais um exemplo do "export + nome da variável=valor":

```
export idade=36
bash mostra_idade
Paulo tem 36 anos
```

Utilizamos o padrão comum, "export + nome da variável=valor", que "Paulo tem 36 anos".

Atenção

Cuidado com os espaços. Por exemplo, se digitarmos `export idade= 38` teremos o seguinte:

```
export idade=38
bash: export: `38`: not a valid identifier
```

E se digitarmos `export idade 38` ? Sem o sinal de "=".

```
export idade=38
bash: export: `38`: not a valid identifier
export idade 38
bash: export: `38`: not a valid identifier
```

Vamos observar o que aconteceu com a variável idade:

```
export idade=38
bash: export: `38`: not a valid identifier
export idade 38
bash: export: `38`: not a valid identifier
echo $idade
```

Ela estará zerada! Quando chamamos só o `export idade` o que estamos fazendo é zerando essa variável, pois estamos pegando o valor da variável e estamos colocando um vazio dentro dela, ela já não tinha valor. Se digitarmos que o valor é `39` , por exemplo, `export idade 39` o que estamos fazendo é pedindo para que uma variável, chamada "39" seja criada.

Atenção! Não podemos criar uma variável que comece por número. Não é um identificador válido.

E se tentássemos, ao em vez de utilizar um número, como "39", tentar com um nome, como "Guilherme"? Por exemplo, criar a variável nome com o valor de "Paulo".

```
echo $nome
Paulo
export nome Guilherme
```

Ele vai falar que tudo está "ok". Mas, lembre, não colocamos o sinal de "=". Então, a variável `$nome` não significa que tenha como valor "Guilherme". Vamos verificar os valores?

```
echo $nome
Paulo
```

A variável nome continua a valer "Paulo". Ela continua valendo a mesma coisa que a variável valia até agora no nosso *shell*. Ela já valia "nome", então, ela continua a valer "nome". E o "Guilherme"? Foi criada uma variável chamada "Guilherme", mas que valor ela possui? Nenhum. Porque não atribuímos a ela valor nenhum, assim, a variável Guilherme continua sem valor.

Repare que é preciso tomar muito cuidado, é necessário que o sinal de "=" esteja na variável. Caso contrário, o que ele faz é manter o valor da variável `nome` com o valor dela. A variável Guilherme que não existia, pois tinha valor vazio, também existe agora como uma variável de ambiente com o valor Guilherme. Então, já que posso colocar com espaço e fazer essa "cacá", eu posso também colocar duas variáveis de uma vez só?

Sim! Vejamos a variável nome e idade juntas:

```
export nome=Ana idade=31
```

Podemos exportar várias variáveis de uma única vez, as vezes é um atalho que podemos utilizar. As vezes temos várias variáveis que queremos exportar de uma vez só e podemos exportá-las como variáveis de ambiente. Agora, se dermos um `echo $nome` e um `echo $idade` ele nos responderá `Ana` e `31`.

```
export nome=Ana idade=31
echo $nome
Ana
echo $idade
31
```

Ambas são variáveis de ambiente. Se digitarmos `bash mostra_idade` teremos `Ana tem 31 anos.`

Repare que o `export` tem esse padrão. O comando se chama `export` e podemos passar para ele diversos parâmetros. Cada parâmetro é interpretado da seguinte maneira, se existe o valor de "=" na variável ele irá ler: `nome da variável=valor da variável`. Se não tem um "=", ele simplesmente pega o valor da variável do *bash* como variável de *shell* e vai exportar como variável de ambiente.

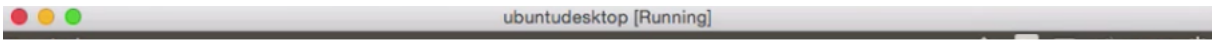
Como provamos que isso está acontecendo? Que ele está criando essas variáveis de ambiente?

Vamos limpar a tela usando o `clear` e pedir para ele mostrar as variáveis de ambiente, lembrando que a palavra em inglês é `enviroment`, então, digitaremos sua abreviação `env`. E `env` irá nos mostrar as variáveis de ambiente quando executarmos o comando solto.

Vamos ver o que acontece se dermos um "Enter" de depois de escrever `env`:

Repare que ele irá nos mostrar um monte de variáveis. E, inclusive, encontraremos `idade=31` .

Se você continuar a rolar a barrinha, poderá ver mais uma série de variáveis. Inclusive, `nome=Ana` .



Repare que tem diversas variáveis de ambiente que já vêm no *shell*. Elas já existem quando iniciamos ele e suas origens são de diversos motivos, por exemplo, temos o `LOGNAME` que mostra o nome do usuário. Vamos falar, mais adiante, sobre diversas dessas variáveis que são padrão.

Antes, vamos ver um pouco mais acerca das características de variáveis de *shell*, as locais, e variáveis de ambiente.

Já falamos que, se criarmos uma variável de ambiente, todos os processos que são executados dentro desse terminal, como filhos deste ambiente, herdarão essa variável de ambiente. E se criarmos um outro terminal? Isto é, rodar um outro terminal?

Repare que temos um outro terminal:

Não é um terminal filho do que estávamos utilizando, é um outro terminal. Será que nele existe a variável nome?

Não! A variável nome não vai existir nesse último terminal que abrimos, não é um terminal filho. Não é como se estivéssemos rodando um *bash* e rodamos um outro *bash* dentro dele. Nós criamos um *bash* separado daquele que nós já tínhamos, então, eles não apresentam um grau de "parentesco", um não é filho do outro.

Esse é um cuidado do dia a dia que temos que ter. E, isso, foi apenas para demostrar e deixar claro que a variável de ambiente só é herdada para os processos filhos daqueles que estão rodando no ambiente atual.

Repassando

O que foi demostrado aqui foram algumas variações do comando `export` . Que pode suportar diversos nomes de variáveis e vai pegar o valor local , de *shell* de cada uma dessas variáveis e aplicar, no caso de `export nome idade` . Ou, podemos escrever da seguinte maneira, `export nome idade=29` nesse caso pegaremos no `nome` o valor de *shell* e aplicaremos como variável de ambiente e `idade=29` pegaremos o valor e simplesmente aplicar como variável de ambiente.

Lembre-se que as variáveis, tanto local quanto de ambiente sempre terão os mesmos valores. No momento em que utilizamos o `export` estamos criando a variável de ambiente ou alterando seu valor e a variável *shell* também passa a ter esse valor. Isso ocorre porque toda vez que tentarmos interpretar a variável `$idade` ele vai pegar a variável de ambiente que terá o valor `29` .

```
export nome idade
export nome idade=29
echo $idade
29
```

E como fazemos para ver todas as variáveis de ambiente? Basta digitar `env` .

E todas as variáveis estarão descritas.

Vamos ver mais de outras variações daqui a pouco.

Mais Variações de env, export e detalhes do bash

Já vimos o comando `export` e o comando `env` .

O que mais que o comando `env` é capaz de fazer?

Vimos que o comando `env` serve para listar todas as variáveis de ambiente, isto é, de enviroment.

Vamos dar uma limpada nesse tela, usando o `clear` .

E se criarmos umas variável de *shell*, `AMBIENTE=teste` ?

Essa máquina, é uma máquina de teste, por isso, criamos uma variável ambiente e colocamos o valor teste nela. Se pegarmos essa variável e rodarmos o `env` , será que aparecerá a palavra `AMBIENTE` listada na nossa tela?

Se verificarmos em nossa lista do `env` , constataremos que `AMBIENTE` não estará em nossa lista. O `env` mostra a lista apenas das variáveis que advêm do ambiente. Está, inclusive, explicíto em `env` , uma vez que seu próprio nome é oriundo de *enviroment*. Não encontraremos `AMBIENTE` , pois não mostrará as variáveis de um *shell*. Estas só são visíveis em um processo específico, na medida em que entramos em um processo novo, como o que estamos realizando ao rodar um `env` , esse processo, não tem acesso as variáveis anteriores, ou seja, elas não estarão mais visualizáveis nesse processo. Vamos limpar a tela com um `clear` .

O que mais o `env` é capaz de fazer?

Ele é capaz de executar comandos em um novo ambiente. Se quisermos criar um novo ambiente, ele fará isso por nós, além disso, nesse novo ambiente ele rodará os comandos que nós quisermos.

Podemos digitar `env echo $nome` e pedir para que ele rode esse comando.

```
env echo nome
Ana
```

Temos o resultado demostrado a cima, pois nome é uma variável de ambiente.

Vamos rodar um novo terminal,

Nesse novo terminal não teremos mais a variável `$nome` , uma vez que, estamos em um novo processo. Agora, podemos criar uma nova variável nome, seu valor será "Guilherme". E, além disso, vamos pedir para o `env` criar um ambiente novo e nesse ambiente pediremos para ele rodar o comando `echo $nome, bem vindo` . Teremos o seguinte:

```
nome=Guilherme
env echo $nome, bem vindo.
Guilherme, bem vindo.
```

Quando rodamos o `env` repare que ele cria um novo ambiente e dentro deste novo cenário, ele não deveria ter acesso as variáveis locais, as variáveis de shell. Por que, então, o `env echo $nome` consegue substituir uma variável de *shell* que é `nome=Guilherme` ? Por que, afinal, isso acontece?

Lembre-se do caso do `ls` ! Uma coisa é o *bash* interpretar todos os parâmetros que serão executados no nosso programa. Isso é fundamental no dia a dia do trabalho do *comand line*. Vamos voltar a esse assunto diversas vezes, teremos que perceber isso no cotidiano e também, por que na prova esse conteúdo cai. Repare que podemos ter a sensação de que ele está criando um ambiente novo, um processo novo e por isso o `$nome` vai ser vazio. Na verdade o que acontece é que o *bash*

primeiro vai falar que `env` é `env`, `echo` é `echo`, `$nome` é `Guilherme`, `bem` é `bem`, , é , , `vindo` é `vindo` e `.` é . . E, por isso, executa o comando `env` passando os parâmetros `Guilherme, bem vindo.`

Lembre-se do `ls`, é o mesmo exemplo! Recorde que tínhamos `padrao=-la` e também tínhamos `ls $padrao`. O `$padrao` é uma variável de *shell* que vai ser interpretada antes de executar o processo `ls`, antes de rodar o comando `ls $padrao`.

```
echo nome
nome=Guilherme
env echo $nome, bem vindo.
Guilherme, bem vindo.
padrao=-la
ls $padrao
```

Ele vai interpretar, vai virar ls -la e, então, é que vai ser executado o `ls -la`. E dentro do `ls -la` não temos acesso a variável `$padrao`. O mesmo procede agora, dentro do comando `echo` que tínhamos inserido antes, ele vai ser executado em um novo *enviroment*, mas não teremos acesso a variável `$nome`.

Vamos executar para verificar o que acontece dando um "Enter" após `ls $padrao`. Ele mostra, mas dentro nós não temos acesso a `$padrao`:

Como podemos provar isso? Nós usaremos um *script* para demonstrar isso. O `bash mostra_idade` que executa o `tem anos`, ocorre mesmo que a variável seja `nome=Guilherme`. Entretanto, quando executamos dentro dele, não temos valor. Dentro desse novo processo não temos acesso a variável. Observe isso acontecendo:

```
bash mostra_idade
tem anos
nome=Guilherme
bash mostra_idade
tem anos
```

Você deve estar se perguntando: ***Para que uso o `env` então?***.

Bom, vamos observar! Podemos pedir para o `env` rodar um comando, `env echo` + "alguma coisa". Vamos retornar ao `bash mostra_idade`, lembre-se que ele nos respondia `tem anos` apesar da variável `$nome` estar lá. Repare:

```
bash mostra_idade
echo $nome
tem anos
```

E se pedirmos para o `env` criar um novo ambiente?

Nesse novo ambiente executaremos o `bash mostra_idade`. O `env` irá criar um novo ambiente e nesse novo ambiente não teremos a variável `nome`, isso ocorre pois esse ambiente é filho do processo atual. Vamos observar o que acontece quando executamos o comando `bash mostra_idade`, processo filho do processo do novo ambiente, teremos como resposta apenas o: `tem anos`. Nada mais será mostrado, pois ele não tem essa variável.

```
env bash mostra_idade
tem anos
```

Para que serve o `env` afinal? Se não tem diferença executar usando ele ou sem ele?

A diferença é que na hora que executamos o `env` , podemos passar, antes do nome do comando, uma série de variáveis que queremos utilizar somente na execução desse comando. Por exemplo `env nome=Guilherme bash mostra_idade` . Digitando isso o que queremos é que se crie um novo ambiente, filho do processo atual, com a variável `nome=Guilherme` como variável de ambiente e, por fim, executamos o `bash mostra_idade` .

```
env nome=Guilherme bash mostra_idade
Guilherme tem anos.
`
```

Veremos, que ele mostrará `Guilherme tem anos` .

Podemos executar o mesmo comando `nome=Guilherme` com `idade=34` :

```
env nome=Guilherme idade=34 bash mostra_idade
Guilherme tem 34 anos.
```

Teremos então, que `Guilherme tem 34 anos` .

E para descobrir o valor da variável nome e Guilherme? Usaremos o `echo` :

```
echo $nome
Guilherme
echo $idade
```

Veremos que ele mostrará um valor vazio. Lembre que o comando `env` nos diz que, somente durante a execução desse comando vamos disponibilizar para este comando as variáveis `$nome` e `$idade` .

Poderíamos usar `env nome=Paulo idade=36 bash mostra_idade` . Ele vai executar `mostra_idade` em um `bash` novo dentro de um ambiente cujo nome é `Paulo` e a idade é `36` . Mandamos ele executar. Lembrando que aquele ambiente que antes estávamos morreu, desapareceu, em meu ambiente atual essas variáveis foram criadas?

Não. pois se digitarmos `echo $nome` e `echo $idade` teremos os valores antigos. Teremos "Guilherme" e "vazio", respectivamente.

```
env nome=Paulo idade=36 bash mostra_idade
echo $nome
Guilherme
echo $idade
```

Repare que o `env` permite que executemos um comando criando variáveis de ambiente temporárias, somente para aquele comando, sem sujar o comando atual. Perceba, é muito fácil sujar o ambiente atual. Para isso basta sair criando variáveis para os diferentes comandos que vão sendo executados, assim, cria-se uma variável, executa um comando, cria-se uma outr

variável, muda o valor e executa mais um comando, cria-se ainda outra variável, troca o valor e executa o comando e por aí vai. Quando percebemos, nosso comando está cheio de variáveis. Mas, perceba, por exemplo, em um caso hipotético, cinco dessas variáveis só são usadas uma única vez quando executamos aquele único comando.

Podemos, chamar o `env` e colocar nele mais de uma variável, a variável 1, 2, 3... Por exemplo, `env nome=Guilherme idade=34` e executamos o comando utilizando o `bash mostra_idade` e aí teremos:

```
env nome=Guilherme idade=34 bash mostra_idade
Guilherme tem 34 anos
```

Podemos falar para o `env` exportar o valor da variável nome e a variável idade `idade=34` . Teremos:

```
env nome idade=34 bash mostra_idade
env: nome: No such file or directory
```

A resposta que recebemos é "não existe esse diretório ou arquivo". Por que não? Se não tem um `=` este `nome` passa a ser o comando que queremos executar. O `env` funciona da seguinte maneira: `env` , sequência de variáveis, `=` e nome do comando. Tem sequências de variáveis como em : `env nome idade=34 bash mostra_idade` ? Não, pois logo tem o nome do comando.

O `env` permite que criemos um *enviroment* novo, que a gente coloque as variáveis dentro do *enviroment* e execute esse *enviroment*. As variáveis só estarão disponibilizadas dentro desse *enviroment*, mas depois que terminarmos a execução desse comando, as variáveis não estarão mais disponíveis. Inclusive se pegarmos o `script` :

Se citarmos no `script` uma nova variável, `profissão=Instrutor` abaixo do que já temos e salvarmos isso, teremos o seguinte:

```
echo $nome tem $idade anos.
profissao=Instrutor
```

Voltando ao terminal podemos executar, `export nome=Guilherme` , `export idade=34` e `bash mostra_idade`

Teremos:

```
export nome=Guilherme
export idade=34
bash mostra_idade
Guilherme tem 34 anos
```

Vamos digitar também `echo $profissao` ? Veremos que a resposta dele será um vazio. Aquela variável do *shell* não foi exportada, pois não faz sentido que aquela variável seja visível.

Vamos retornar ao editor:

Criamos uma variável de processo, de *shell*, então, ela está disponibilizada apenas na execução do `script`. Quando acaba, acabam também as variáveis do *shell*.

Então, e se mudarmos a variável nome, no editor para Paulo e salvar, o que acontece?

```
echo $nome tem $idade anos.
profissao=Instrutor
nome=Paulo
```

E se executarmos novamente o script no terminal? Digitaremos `bash mostra_idade` e teremos o seguinte:

```
bash mostra_idade
Guilherme tem 34 anos.
```

Ele mostra `Guilherme`, mas se tentarmos mostrar a variável `nome`, através do `echo`, teremos, novamente `Guilherme:

```
bash mostra_idade
Guilherme tem 34 anos.
echo $nome
Guilherme
```

Isso ocorre pois ele apenas alterou isso no `script`. Mas, será que houve mesmo alguma alteração?

Vamos no editor e digitamos `echo O novo nome e $nome e a profissao $profissao`. Teremos o seguinte:

```
echo $nome tem $idade anos.
profissao=Instrutor
nome=Paulo
echo O novo nome e $nome e a profissao $profissao
```

Vamos salvar isso e vamos executar de novo o `bash mostra_idade`:

```
bash mostra_idade
Guilherme tem 34 anos.
O novo nome e Paulo e a profissao Instrutor.
```

Temos que o `O novo nome e Paulo e a profissao Instrutor`. E se pedirmos para ele o `echo $nome` e o `echo $´profissao`?

```
bash mostra_idade
Guilherme tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
echo $nome
Guilherme
echo $profissao
```

Podemos observar que quando pedimos pelo `echo $nome` e o `echo $profissao` ele devolve os valores antigos. Tudo que *setamos* foram as variáveis de *shell* que só estão visíveis no processo do editor:

Vamos no editor! Se quero transformar essas variáveis em variáveis de ambiente o que podemos fazer? Temos que ao digitar `export` ele pode ser acompanhado de "alguma coisa", por exemplo, `export nome` ou, como poderíamos fazer direto, `export profissao=Instrutor` . As duas maneiras são possíveis. Se usamos o `export nome` ele já pega o valor da variável `nome` e exporta para a variável de ambiente.

Teremos:

```
echo $nome tem $idade anos.
profissao=Instrutor
nome=Paulo
echo O novo nome e $nome e a profissao $profissao
export nome
```

Vamos fazer um teste agora, vamos limpar a tela usando o `clear` e vamos ver nossas variáveis. Vamos executar o `bash mostra_idade` e teremos:

```
echo $nome
Guilherme
echo $profissao

echo $idade
34
bash mostra_idade
Guilherme tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
```

Demos um `export` no nome e na profissão, o nome e a profissão deveriam vir, respectivamente, como variável de ambiente `Paulo` e `Instrutor` . Vamos ver se isso acontece utilizando o `echo $nome` e `echo $profissao` :

```
echo $nome
Guilherme
echo $profissao
```

Repare que não temos o que escrevemos lá no `export` , temos "Guilheme" para nome e "vazio" para profissão.

O que aconteceu? Ele ignorou o `export` ?

O `export` não deveria setar a variável que era uma variável de *shell* somente desse processo para uma variável de ambiente que é para o processo atual e para os processos filhos dela?

Sim. isso procede!

Mas, atenção:

Se exporto uma variável ela vira uma variável de ambiente, para ela e para o processo atual. O processo do editor está executando o processo do script e os processos filhos dele. Todos os comandos que executarmos dentro dele, os filhos do processo atual vão herdar as variáveis de ambiente, de *enviroment*.

Uma vez que matamos esse processo e retornamos ao processo pai ou mãe que está no nosso terminal, eles não estabelecem nenhuma relação. A herança é apenas para os processos filhos!

Repare que todas as variáveis de ambiente são herdadas pelo comando `bash mostra_idade`. O `mostra_idade` roda tudo o que definimos no ambiente dele e isso é herdado pelos filhos do `mostra_idade`, mas na hora que os filhos do `mostra_idade` acabarem e na hora que o `mostra_idade` acabar, tanto faz o que ele tinha setado, pois, o que ele tinha setado fica apenas para ele.

A grande sacada disso é que um *script* não seja capaz de alterar variáveis de ambiente do pai e da mãe, isso poderia ser perigoso! Imagine um caso, estamos no terminal com algumas variáveis que garantem algum tipo de segurança e temos certeza de como as variáveis funcionam e ao executar um script alteramos através de um `export` uma variável que não queríamos que fosse alterada, isso pode causar problemas. O `export`, portanto, só setará para os filhos.

Então, `export` só funciona para o *shell* atual e seus filhos, assim, quando retornamos para quem chamou o processo atual, não tem diferença o que foi setado.

Limpamos nossa tela, dando um `clear`.

Vamos realizar um outro teste. Temos a variável `nome`, que possui o valor `Guilherme` e digitamos também o `bash mostra_idade` que irá executar o `Guilherme tem 34 anos`. Teremos o seguinte:

```
echo $nome
Guilherme
bash mostra_idade
Guilherme tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
```

Quando executamos o `mostra_idade` ele irá nos mostrar `Guilherme tem 34 anos`,pois já demos um `export`, o `export nome=Guilherme` e executamos o `bash mostra_idade`. Se dermos um `env` veremos que a variável `nome` estará na nossa lista de variáveis de ambiente:

Vamos dar um `clear` para limpar a tela. Bom, concluímos que `nome` é uma variável de ambiente. E se falarmos agora que a variável nome, é Paulo, `nome=Paulo`?

A variável Paulo e Guilherme são variáveis diferentes? Ou, uma única variável que está sendo exportada e que agora trocamos o valor dela?

Vamos fazer o teste:

```
nome=Paulo
bash mostra_idade
Paulo tem 34 anos
O novo nome e Paulo e a profissao Instrutor
```

Repare, o que o `export` faz não é simplesmente uma coisa pontual, `export nome=Guilherme`. O `export` exportou uma variável de *shell* para uma variável de ambiente, mas, não é apenas isso, ele também fala que a partir de agora a variável chamada nome é de ambiente, não importa o valor que ela assuma.

Poderíamos, por exemplo, colocar o nome "Ana" e quando executarmos o `bash mostra_idade` teremos:

```
nome=Paulo
bash mostra_idade
Paulo tem 34 anos
O novo nome e Paulo e a profissao Instrutor
nome=Ana
bash mostra_idade
Ana tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
```

E o mesmo ocorrerá se seguirmos alterando o nome, por exemplo, "Carlos":

```
nome=Paulo
bash mostra_idade
Paulo tem 34 anos
O novo nome e Paulo e a profissao Instrutor
nome=Ana
bash mostra_idade
Ana tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
nome=Carlos
bash mostra_idade
Carlos tem 34 anos
O novo nome e Paulo e  a profissao Instrutor
```

E se alteramos, ainda, pelo nome Lucia? Teremos a mesma coisa:

```
nome=Paulo
bash mostra_idade
Paulo tem 34 anos
O novo nome e Paulo e a profissao Instrutor
nome=Ana
bash mostra_idade
Ana tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
nome=Carlos
bash mostra_idade
Carlos tem 34 anos
O novo nome e Paulo e a profissao Instrutor
nome=Lucia
bash mostra_idade
Lucia tem 34 anos
O novo instrutor e Paulo e a profissao Instrutor
```

Vamos no `help export` e ver o que está escrito! Teremos o seguinte:

Na terceira linha já está escrito que ele seta atributos de exportação para variáveis do *shell*. Isto é, ele marca cada um dos nomes que passamos como exportação automática. A partir do momento que demos um `export` não precisamos mais dar `export` no *shell* atual porque o *shell* atual tem essa variável de ambiente.

Se tenho essas variáveis, e se dermos um `env` encontraremos o `nome=Lucia` :

Repare que não é que existem duas variáveis com valores distintos, uma variável de *shell* e uma de ambiente, existem variáveis no *shell*, dezenas delas. Através do comando `export` nós marcamos as variáveis como variáveis de ambiente, de *enviroment*. A partir desse instante, as variáveis desse processo serão visíveis tanto pelo processo atual, quanto pelos processos filhos e filhas.

A partir de agora, se mudarmos o valor da variável nesse processo, ela também será alterada e disponibilizada dessa maneira para os processos filhos e filhas. A variável é uma só, a única coisa é que ela está marcada como exportável, como uma variável de ambiente.

Vamos fechar o terminal que estávamos usando até o momento e abrir um novo. Esse novo terminal não possui nenhuma das variáveis que o outro tinha. Se quiser verificar, basta digitar `echo $nome $idade` e observar que não teremos retorno. Vamos simplesmente digitar `env nome=Guilherme idade=34 bash mostra_idade` . Teremos:

```
echo $nome $idade

env nome=Guilherme idade=34 bash mostra_idade
Guilherme tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
```

E se usarmos o cifrão para definir a variável, isto é `env $nome=Paulo $idade=35 bash mostra_idade` ?

Veremos que:

```
echo $nome $idade

env nome=Guilherme idade=34 bash mostra_idade
Guilherme tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
env $nome=Paulo $idade=35 bash mostr_idade
tem anos.
O novo nome e Paulo e a profissao Instrutor
```

Teremos como resposta apenas `tem anos` . Ele ignorou, mas por que?

Lembre, o `$nome` significa que é para o nosso *shell* interpretar isso. O que ele está executando, na verdade é o comando: `env Guilherme=Paulo 34=35 bash mostra_idade` . Se executarmos isso teremos `tem anos` .

Lembre, o cifrão é interpretado antes da execução do comando, por isso temos que tomar cuidado para não colocar o cifrão na frente. Repare que isso revela outro detalhe importante que já foi comentado anteriormente, criar uma variável começando com número não é possível. Se não, teremos o seguinte:

```
34=35
34-35: command not found
```

Repare que o `env` funciona se digitamos `env 34=35 bash mostra_idade` , pois conseguimos criar uma variável, mas esse criar é bem entre aspas, pois, ela começa com um número. Como observamos, digitar apenas o `34=35` solto, não é possível.

```
env 34=35 bash mostra_idade
tem anos
O novo nome e Paulo e a profissao Instrutor
34=35
34-35: command not found
```

Será que ele criou mesmo uma variável com nome inválido no caso do `env 34=35 bash mostra_idade` ? Ou ele simplesmente engoliu o erro e não falou nada? Vamos testar isso!

Vamos digitar `env 34=35 env` , o último `env` é para que justamente este comando seja executado. Ele executa o comando `env` e mostra até a variável `34=35` .

Vamos testar essa variável? Vamos dar um `clear` e vamos digitar `env 34=35 echo O numero e $34` . Teremos:

```
env 34=35 echo O numero e $34
O numero e 4
```

Bom, não funcionou! Temos o `O numero e 4` . Repare que até podemos criar a variável `34=35` , aquela variável com número que o *bash* não estava deixando, mas na hora de interpretar o resultado teremos um problema.

Você pode imaginar que ele interpreta apenas o `3` , mas esses detalhes já são tão finos que a prova não vai chegar a eles.

Vamos compreender, vamos digitar o seguinte: `env 1=guilherme echo o nome e $1` . Ele imprimirá apenas a palavra "Guilherme". Executaremos e teremos:

```
env 34=35 bash mostra_idade
O numero e 4
env 1=guilherme echo o nome e $1
o nome e
```

Ele imprime apenas `o nome e` e fica quieto, não fala nada. O `1` é uma variável especial, esse tipo de variável veremos mais adiante. Isso também ocorre com `$3` e o `$34` . Tudo o que é cifrão de um número é uma palavra diferenciada, isto é, uma variável especial. Por esse motivo não podemos sair criando variáveis especiais como essa, caso contrário, ele ficaria confuso.

Nós tentamos criar usando o `1=guilherme` e o `$1` , mas repare que ele ignorou `$1` . Isso indica que temos que tomar alguns cuidados.

Veremos as variáveis especiais que começam com número mais adiante, quando formos falar de `script` . No nosso caso presente, temos diversas variáveis especiais, mas que não começam com número, que não são usadas para execução de `script` . Entretanto, são variáveis importantes do nosso dia.

Então, já nos debruçamos sobre alguns detalhes do `env` , onde, " `env` +algum valor" é que executa o comando. No caso do `bash` , especificamente, vamos observar outro meio que não apenas falar `env nome=Joao idade=38` e executar isso usando o `bash mostra_idade` . No comando `bash` , especificamente, e em suas famílias e derivados podemos, no nosso shell não colocar a palavra `env` que, igualmente, ele automaticamente irá criar as duas variáveis `nome=Joao` e `idade=38` . Quando

executamos um comando o que fazemos é criar variáveis com essas variáveis. O *bash* vai utilizar essas variáveis no `mostra_idade`. Se executar teremos:

```
nome=Joao idade=38 bash mostra_idade
Joao tem 38 anos.
O novo nome e Paulo e a profissao Instrutor
```

Vamos executar que `nome=Lucia idade=22 bash mostra_idade`:

```
nome=Lucia idade=22 bash mostra_idade
Lucia tem 22 anos
O novo nome e Paulo e a profissao Instrutor
```

Ele mostrará que Lucia tem 22 anos. Repare que no *bash* e na família de *shells* derivada do *bash* não precisamos colocar a palavra `env` antes, é opcional. O formato de um comando, a sintaxe do comando, quando utilizamos um *bash* pode ser simplesmente os parâmetros. No caso `nome=Lucia` e `idade=22`. O *bash* funciona com a sintaxe de um comando, que não necessariamente precisa ter a ordem "nome + opções + argumento", relembrando que a ordem de opções e argumentos depende, na verdade do programa, mas essa é normalmente a norma padrão.

Quando estamos em um *bash* também podemos executar da seguinte maneira:

variáveis=valores nome opções argumentos

A variável "variáveis=valores" vai ter essa variável atribuída somente durante a execução desse comando. É equivalente a criar um novo `enviroment` com esse valor que após executado morre.

Vamos testar?

Vamos dar um `echo $nome`, que está até agora vazio e executamos o `nome=Guilherme idade=34 bash mostra_idade`. Ele executa `Guilherme tem 34 anos` e a variável nome segue vazia:

```
nome=Guilheme idade=34 bash mostra_idade
Guilherme tem 34 anos.
O novo nome e Paulo e a proissao Instrutor
```

A palavra `env` é opcional. Na maior parte das vezes ela não aparece. Veremos direto "nome=valor" e o comando que se quer passar, as opções e os argumentos.

Com isso vimos diversas variações de `env`, `export`, de variáveis de *shell*, como marcar variáveis de *shell* com `export` para visualizar com `env`, setar variáveis de ambiente com `env` para logo de cara executar um comando novo nesse ambiente também novo e omitir a palavra `env` e criar um ambiente novo para execução daquele comando especificamente que funciona no *bash* e seus derivados de *shell*.

Padrões para nomes de variáveis

Vamos olhar no *script* o que foi criado:

Agora, vamos rodar o *script* com um nome completo, por exemplo, vamos escrever no terminal `nome=Guilherme Silveira` . Teremos a seguinte resposta:

```
> nome=Guilherme Silveira
Silveira: command not found
```

Uma das maneiras de executar comandos no *bash* é: `nome de uma variável=valor da variável e nome do comando` . Observando a ordem que a nossa linha de comando deve ter, como será que Silveira foi interpretado? Ele interpretou que `nome` é o nome da variável, `Guilherme` é o valor da variável e `Silveira` foi interpretado como o nome do comando.

Entender a sintaxe no *shell,* no caso, no *bash* é extremamente importante, assim, evitamos cair em erros como esse. Podemos cometer o equívoco de acreditar que estamos executando uma coisa quando na verdade executamos outra. Vamos fazer outro tese:

```
nome=Guilherme Silveira bash mostra_idade
Silveira: command not found
```

Continuamos com um erro, pois, existe o espaço entre `Guilherme Silveira` . Como fazemos para que o espaço entre o nome e o sobrenome sejam ignorados?

Existem diversas maneiras de fazer isso, nesse momento nós vamos utilizar uma dessas formas. E mais adiante nesse curso falaremos sobre as demais interpretações de como o *shell* lê o comando que está escrito para que possamos executá-lo. Isto é, vamos em outro momento passar por diversos outros casos.

O primeiro caso que vamos observar é o das aspas. Podemos utilizar uma aspas simples no valor que queremos atribuir a nossa variável, as aspas simples servirão como uma marcação para que o espaço não seja ignorado e não nos atrapalhe na sequência, na leitura do que queremos passar. Teremos, utilizando as aspas, o nome completo "Guilherme Silveira":

```
> nome=`Guilherme Silveira`
echo $nome
Guilherme Silveira
```

Veremos, mais adiante, que as aspas duplas podem ser utilizadas também para criar espaço entre o que queremos inserir como valor da variável.

Repare que se digitarmos o `echo $nome` teremos como resposta o `Guilherme Silveira` .Observe que a variável não é exportada, então, se quisermos exportá-la temos que digitar `export nome=` Guilherme Silveira``.

Se pedirmos `bash mostra_idade` a resposta será:

```
> bash mostra_idade
Guilherme Silveira tem anos.
O novo nome e Paulo e a profissao Instrutor
```

Podemos fazer também, `nome=` Paulo Silveira `bash mostra_idade` :

```
> nome=`Paulo Silveira` bash mostra_idade
Paulo Silveira tem anos
```

```
O novo nome e Paulo e a profissao Instrutor
```

Atenção! É preciso ter cuidado na hora em que criamos as variáveis. Não apenas o espaço depois ou antes do "=" é perigoso como também o espaço entre o valor e o comando. Se o espaço aparece, por exemplo, depois do "=" em `nome= Guilherme` o que estamos fazendo é criar uma variável vazia cujo comando é `Guilherme` . E essa variável não existe:

```
> nome= Guilherme
Guilherme: command not found
```

Isso ocorre devido ao padrão que temos que obedecer: o de não inserir espaços. Portanto, não podemos inserir os espaços, mas se queremos que eles estejam explícitos no texto que inserimos, temos que acrescentar aspas simples. Existem diversas maneiras de como lidar com espaços, `Strings` e `Escapes` , porém, veremos isso mais adiante.

Estamos utilizando a variável nome no `script` . Antes, vamos ver as variáveis de ambiente digitando no terminal `env` :

```
<img src="https://s3.amazonaws.com/caelum-online-public/essentiallinux3/essential2_4/essenlinux2_4_1
```

Podemos observar diversas variáveis, `UPSTART JOB=unity7` , `DISPLAY=:0` , `LESSCLOSE=/usr/bin/lesspipe %s %s` , `LOGNAME=guilherme` e etc. O que elas todas tem em comum?

Todas estão em letras maiúsculas. Existe um padrão para os nomes de variáveis de ambiente, que é deixá-las com letras todas maiúsculas. Não é obrigatório, até o momento estávamos usando as variáveis com letras minúsculas. Mas, é um padrão adotado.

Assim, quando lermos nosso *script*, por exemplo, na imagem abaixo, fica mais fácil diferenciar o que é uma variável e o que não é, não é mesmo? Não.

E se alterarmos as variáveis por letras maiúsculas?

Agora sim, simplesmente de olhar, sabemos diferenciar o que é variável do que não é. Temos uma definição de um padrão, onde as variáveis são digitadas usando as letras maiúsculas para identificar visualmente e facilmente o que elas são. Usaremos esse padrão quando quisermos executar uma variável:

```
NOME=Guilherme bash mostra_idade
Guilherme tem anos.
O novo nome e Paulo e a profissao Instrutor
```

Podemos ver que a idade está vazia, vamos completar a idade usando o padrão que acabamos de aprender:

```
NOME=Guilherme IDADE=34 bash mostra_idade
Guilherme tem 34 anos.
O novo nome e Paulo e a profissao Instrutor
```

Apesar de usarmos as letras maiúsculas ainda existem algumas diferenças entre o padrão que eles utilizam e o que nós estamos utilizando.

Vamos voltar ao `env` para ver se reparamos essas diferenças:

Tudo está escrito em maiúsculo, mas também em inglês, observe o caso do `UPSTART INSTANCE`. Então, tentaremos assumir o mesmo padrão para as variáveis de ambiente de *shell*. Modificaremos os nomes, por nomenclaturas do inglês, voltamos no *script* e alteraremos `NOME` por `NAME`, `IDADE` por `AGE`, `PROFISSAO` por `JOB` e salvaremos essas modificações. Por fim, teremos no script o seguinte:

Observe que os nomes das variáveis estão em inglês. Agora, voltando ao terminal, digitaremos as variáveis conforme esses padrões, usando maiúsculas e nomes em inglês:

```
> NAME=Guilherme AGE=34 bash mostra_idade
Guilherme tem 34 anos
O novo nome e Paulo e a profissao Instrutor
```

Por fim, estamos utilizando o mesmo padrão que eles.

Como nosso *script* está ficando bastante avançado, vamos colocar nele o endereço, que será o `echo Eu moro em $HOME`:

```
echo $NAME tema $AGE anos.
export JOB=Instrutor
NAME=Paulo
echo O novo nome e $NAME e a profissao $JOB export NAME
echo Eu moro em $HOME
```

Vamos voltar ao terminal e executar o `script`:

```
> NAME=Guilherme AGE=34 bash mostra_idade
Guilherme tem 34 anos
O novo nome e Pualo e a profissao Instrutor
Eu moro em /home/guilherme
```

Repare que apesar de ter esquecido de acrescentar a variável `HOME` de qualquer maneira nosso *bash* já mostra para nós o `Eu moro em /home/guilherme`. Mas, isso está errado! Eu não moro em `/home/guilherme`. Isso significa que utilizamos uma variável de ambiente que já existia.

Se retornarmos ao `env` poderemos verificar que em algum lugar teremos a variável `HOME=/home/guilherme`:

Nós utilizamos uma variável de ambiente que já existia. Repare que o padrão recomendado é que, se é uma variável de ambiente ela é em letra maiúscula e em inglês, então, sabemos que as variáveis que entraram até agora são as variáveis que chegaram em letra maiúscula.

Isso no permite que criemos e utilizemos variáveis em letra minúscula a vontade. Assim, as variáveis que nós mesmos criarmos, serão em letra minúscula. Exceto, claro, as variáveis que exportarmos, que utilizaremos, igual, com letra maiúscula.

Vamos remover todo o exemplo que tínhamos escrito no *script* deixando apenas o `NAME` e `AGE` que são as variáveis que queremos receber no nosso ambiente. Ficaremos com:

```
echo $NAME tem $AGE anos.
```

Se queremos criar uma variável temporária, por exemplo, `job=Instrutor` deixaremos ela em letra minúscula, uma vez que ela é temporária, ela só estará aqui para o processo, para o *shell*. Utilizando a letra minúscula para essas variáveis não precisamos ter o receio de que isso possa acarretar problemas com outras variáveis. Como posso afirmar isso? Não tem nenhuma variável de ambiente nos outros *scripts* que seja com letra minúscula e com o mesmo título, uma vez que, todas as variáveis de ambiente, por padrão, adotam o maiúsculo.

Nos assumiremos esse padrão de agora em diante, porém isso não significa que ele é sempre uma regra padrão. Podem ocorrer variáveis escritas utilizando as letras minúsculas, assim como, nós mesmos já escrevemos antes.

E se queremos exportar uma variável, criar, executar alguma coisa? Então, temos nome, idade e endereço. Vamos utilizar, no *script*, o endereço de verdade:

```
echo $NAME tem $AGE anos.
job=Instrutor
export ADRESS=`Rua Vergueiro, 3185`
```

A partir da linha `export ADRESS=` Rua Vergueiro, 3185 essa variável de ambiente poderia ser utilizada no atual processo e nos processos filhos do atual processo.

Usaremos o minúsculo quando a variável for uma variável de *shell* e maiúsculo quando ela for uma variável de ambiente.

Lembrando que esse é um padrão que adotamos, o que não significa que não possa funcionar de outras maneiras, pois, poderíamos criar qualquer uma das duas.

Repare que no ambiente só temos variáveis em maiúsculo, justamente, por que esse foi o padrão adotado para variáveis de ambiente. Repare em nome:

Nós é que quebramos esse padrão ao adicionar uma variável de ambiente com letra minúscula.

Lembre-se que a imagem é do `env` que nos mostra apenas as variáveis de ambiente.

Essa é a importância do maiúsculo e do minúsculo para distinguir, apenas de olhar, qual a característica de cada variável. Se ela é uma variável de ambiente estará em maiúscula, se for uma variável utilizada no script, ela será em minúsculo.

Um último exemplo de variável é usar um nome completo, um `FULL NAME` . Então, estamos no *script* e completaremos o `$NAME` , que ficará `$FULL NAME` :

```
ECHO $FULL NAME, voce tem $AGE anos
job=Instrutor
```

```
expor ADDRESS=`Rua Vergueiro, 3185`.
```

Damos um `clear` na tela do terminal. Digitaremos `FULL NAME=` Guilherme Silveira `, AGE=34 e bash mostra_idade`. Se dermos um "Enter" nisso teremos a seguinte resposta:

```
FULL NAME=`Guilherme Silveira` AGE=34 bash mostra_idade
FULL: command not found
```

Temos um erro! Mas, por que?!

O problema está no espaço do `FULL NAME`, o espaço mostra que `NAME` seria um comando e ao buscar esse comando para executar ele constata que não o encontra e, por isso, temos como resposta: `command not found`. Concluímos, a partir disso, que não faz sentido, no nome de uma variável, existir um espaço.

O padrão que se usa no `env`, como podemos reparar na imagem abaixo é que para remeter a um espaço utilizaremos um *underline*, "_". Isto é, esse *underline* ficará no lugar onde gostaríamos que ocorresse um espaço. Como podemos observar pelas variáveis da imagem abaixo, como o `UPSTART_INSTANCE:

Então, no editor, o padrão que deveríamos ter utilizado é o do *underline*. Podemos modificar o `FULL NAME` por `FULL_NAME`. Teremos:

```
ECHO $FULL_NAME, voce tem $AGE anos
job=Instrutor
expor ADDRESS=`Rua Vergueiro, 3185`.
```

Vamos tentar testar novamente no terminal. Digitaremos `FULL_NAME=` Guilherme Silveira `, AGE=34 e bash mostra_idade` e damos um "Enter".

```
FULL_NAME=`Guilherme Silveira` AGE=34 bash mostra_idade
Guilherme Silveira, voce tem 34 anos.
```

Funcionou! Mas, repare que tem uma vírgula depois de `Guilherme Silveira`. Vamos retirar ela do nosso editor, apagando a vírgula depois de `FULL_NAME`:

```
ECHO $FULL_NAMEvoce tem $AGE anos
job=Instrutor
expor ADDRESS=`Rua Vergueiro, 3185`.
```

Salvamos e vamos testar novamente:

```
FULL_NAME=`Guilherme Silveira` AGE=34 bash mostra_idade
tem 34 anos.
```

Ops! Novamente, o resultado não é o que esperávamos, temos apenas `tem 34 anos`. A vírgula definitivamente sumiu e com ela também o nome.

O que deu errado?

Vamos voltar no editor e observar uma coisa: retiramos o espaço entre `FULL_NAME` e `voce`.

```
ECHO $FULL_NAMEvoce tem $AGE anos
job=Instrutor
expor ADDRESS=`Rua Vergueiro, 3185`.
```

Parece que estamos buscando uma variável que se chamaria `FULL_NAMEvoce` que não será encontrada, mas a variável que queríamos ter setado era apenas `FULL_NAME`. Como fazemos para concertar isso? Usamos as chaves, digitaremos `${FULL_NAME}`. A partir de agora temos um texto solto. Salvamos nossas alterações. Vamos observar como ficou nosso script:

```
ECHO ${FULL_NAME}voce tem $AGE anos
job=Instrutor
expor ADDRESS=`Rua Vergueiro, 3185`.
```

Agora, vamos executar, novamente no terminal:

```
> FULL_NAME=`Guilherme Silveira` AGE=34 bash mostra_idade
Guilherme Silveiravoce tem 34 anos.
```

Repare que temos o nome de uma variável colado ao próximo carácter no nosso script em `${FULL_NAME}voce`. Se não é um espaço que está separando as coisas, corremos o risco de que ele interprete a palavra inteira como o nome da variável, assim, podemos acabar nos perdendo. Portanto, é uma boa prática utilizar as chaves quando estamos utilizando as variáveis. É sempre recomendável utilizar as chaves, pois nós não temos certeza se vai funcionar ou não, por exemplo, podemos digitar uma vírgula depois de `AGE` e testar para ver como fica:

```
ECHO ${FULL_NAME}voce tem $AGE, anos
job=Instrutor
expor ADDRESS=`Rua Vergueiro, 3185`.
```

Vamos no terminal e ver o que acontece:

```
> FULL_NAME=`Guilherme Silveira` AGE=34 bash mostra_idade
Guilherme Silveiravoce  tem 34, anos.
```

Repare que o 34 tem a vírgula que acrescentamos no *script*. Quando o próximo carácter é uma letra, isto é, um identificador válido para o nome de uma variável, o *bash* se perde. Esse foi o caso do `voce` por este motivo utilizamos as chaves em `{FULL_NAME}`, mas, no caso da vírgula depois do `AGE` ele não se perde, pois, não é uma letra.

Bom, então se as vezes ele se perde e as vezes não, se depende do caso, se temos que tomar muito cuidado, se alguém digita algo errado pode acabar quebrando o que já fizemos...

Repare! Ficamos bastante suscetíveis ao erro na utilização de variáveis. É por isso que, na prática, é recomendável, independente se é necessário ou não, que se use o cifrão com as chaves para referenciar as variáveis.

```
ECHO ${FULL_NAME}voce tem ${AGE} anos
job=Instrutor
expor ADDRESS=`Rua Vergueiro, 3185`.
```

Vamos salvar isso e executar mais uma vez:

```
> FULL_NAME=`Guilherme Silveira` AGE=34 bash mostra_idade
Guilherme Silveiravoce tem 34 anos.
```

Observe que tanto faz se o próximo carácter do nosso script é uma letra ou uma pontuação, e se alguém apagar o espaço sem querer que existe entre `${AGE}` e `anos` continuará funcionando normalmente:

```
> FULL_NAME=`Guilherme Silveira` AGE=34 bash mostra_idade
Guilherme Silveiravoce tem 34anos.
```

Para finalizar vamos só deixar um espaço entre o `${FULLNAME}` e o `voce` :

```
ECHO ${FULL_NAME} voce tem ${AGE} anos
job=Instrutor
expor ADDRESS=`Rua Vergueiro, 3185`.
```

Relembrando...

As chaves são importantes para definirmos o nome de uma variável. Além disso, utilizaremos o *underline* e no lugar de um espaço quando queremos definir o nome de uma variável que tenha espaço. Vimos também que é comum, para o nome de uma variável termos todas as letras em maiúsculo para as variáveis de ambiente e todas as letras em minúsculo para as variáveis que pertencem a este *shell*. E assim, saberemos qual é qual.

Tentaremos seguir esse padrão no nosso dia a dia.

Removendo variáveis com o `export` e o `unset`

Já vimos como criar variáveis de *shell* e variáveis de *enviroment*, isto é, de ambiente. Agora, vamos verificar como conseguimos "descriamos" essas variáveis, isto é: Como removemos elas?

Podemos criar uma variável, por exemplo, que fala que o ambiente da máquina que estamos trabalho é de desenvolvimento, uma `AMBIENTE=desenvolvimento` e os programas usam essa variável para definir que tipo de coisas eles devem fazer.

Vamos setar essa variável. E se queremos verificar o valor dela? Utilizaremos o `echo $AMBIENTE` :

```
> AMBIENTE=desenvolvimento
> echo $AMBIENTE
desenvolvimento
```

E como podemos zerar essa variável?

Podemos simplesmente falar que `AMBIENTE=` , digitando isso o que estamos dizendo é que `AMBIENTE` é igual a vazio.

A partir de agora, se dermos um `echo $AMBIENTE` teremos como resposta vazio.

Para uma variável de *shell* basta atribuir a ela um valor vazio e ela estará vazia.

Agora, como fazemos isso para atribuir um valor vazio a uma variável de ambiente?

Temos uma variável de ambiente que chamaremos de `AMBIENTE` . Faremos um `export AMBIENTE=desenvolvimento` .

Para verificar que ela é uma variável de ambiente podemos dar um `env` e buscá-la em nossa lista:

Se só citarmos que `AMBIENTE` é igual a vazio, teremos, `AMBIENTE=` e se fizermos o `echo $AMBIENTE` , teremos um vazio de novo. E se dermos um `env` e procurarmos nossa variável `AMBIENTE` na lista de variáveis de ambiente verificaremos que ela ainda existe!

Mas, ela está com um valor vazio! Existir com um valor vazio é diferente de não existir. Se desejo realmente remover a existência dessa variável, aí, é outra história.

Vamos dar uma olhada na ajuda do export? Vamos digitar `help export` e vamos observar a a opção `-n` :

Observamos que a descrição da opção `-n` é que ela é capaz de remover a propriedade de exportação da variável, então, a variável deixará de ser exportada.

Quando usamos o `export` o que estamos falando é para que se passe a exportar a variável, mas quando falamos `export -n` estamos falando para deixar de exportá-la.

Vamos dar um `clear` e fazer o passo a passo:

```
> AMBIENTE=desenvolvimento
> export AMBIENTE
```

Vamos verificar se `AMBIENTE` está em nossa lista digitando um `env` . `AMBIENTE` está em nossa lista :

Vamos limpar a tela através do `clear` e vamos digitar `export -n AMBIENTE` :

```
> export -n AMBIENTE
```

Agora, se dermos um `env` verificaremos que `AMBIENTE` não estará mais em nossa tela, ela realmente terá desaparecido de nossa lista de variáveis de ambiente. Porém, se buscarmos a variável através do `echo AMBIENTE` , teremos:

```
echo AMBIENTE
desenvolvimento
```

A variável de `AMBIENTE` continuará existindo com o valor desenvolvimento, isto é, ela seguirá existindo no *shell*, só deixamos de exportá-la. O que nós fizemos ao digitar o `export -n` é que deixamos de exportá-la, não falamos que era para remover a definição da variável, isto é, não pedimos para "desetar" a variável.

O que já fizemos foi colocar um valor na variável, isto é, nós *setamos* ela e agora nós queremos que ela seja "desetada" definitivamente. Isto é, o que desejamos é que a variável tenha seu valor removido, para isso, podemos usar o `unset` . Vamos digitar o `unset` e escrever na sequência um `echo $AMBIENTE` para verificar o que ocorre:

```
unset AMBIENTE
echo $AMBIENTE
```

Agora sim! A variável desapareceu, mesmo do *shell* atual. E se a variável fosse uma variável de ambiente?

```
export AMBIENTE=desenvolvimento
```

Se dermos um `env` , verificaremos `AMBIENTE=desenvolvimento` em nossa lista:

Vamos dar um `unset AMBIENTE` e se dermos em seguida um `echo $AMBIENTE` poderemos observar que não teremos nenhum resultado.

```
> unset AMBIENTE
> echo $AMBIENTE
```

Se dermos um `env` verificaremos que não teremos o `AMBIENTE=desenvolvimento` em nossa lista de variáveis de ambiente.

O `unset` remove do *shell* e *unseta*, ou seja, ele também faz um `export -n` da variável ambiente.

Retomando: O `unset` faz duas coisas, ele vai remover a variável do *shell* e dar um `export -n` . Se dermos apenas um `export -n` na variável estamos apenas tirando-a da nossa lista de variáveis a serem exportadas, mas ela continuará existindo como uma variável do *shell* atual.

Então, vimos que são diversas as maneiras de remover uma variável que existe no sistema.