LAB - HABILIDADES EM LINHA DE COMANDO

4.1 Introdução

Este é o laboratório 4: Noções básicas de linha de comando. Ao realizar este laboratório, vocês aprenderão como usar os recursos básicos do shell.

Neste laboratório, você executará as seguintes tarefas:

- Explorar recursos do Bash
- Usar variáveis shell
- Entender como usar globbing
- Ser capaz de fazer uso de aspas

4.2 Arquivos e Diretórios

Nesta tarefa, acessaremos a CLI (Command Line Interface, interface de linha de comando) do Linux para explorar como executar comandos básicos e o que afeta a maneira como eles podem ser executados.

A maioria dos usuários provavelmente está mais familiarizada com a forma como os comandos são executados usando uma interface gráfica do usuário (GUI). Portanto, essa tarefa provavelmente apresentará alguns novos conceitos para você, caso você não tenha trabalhado anteriormente com uma CLI. Para usar uma CLI, você precisará digitar o comando que deseja executar.

A janela onde você digitará seu comando é conhecida como um aplicativo de emulador de terminal. Dentro da janela do Terminal, o sistema está exibindo um prompt, que atualmente contém um prompt seguido por um cursor piscando:

```
sysadmin@localhost:~$
```

Lembre-se: Você pode precisar pressionar **Enter** na janela para exibir o prompt.

O prompt informa que você é usuário `sysadmin`; o host ou computador que você está usando: `localhost`; e o diretório onde você está: `~`, que representa seu diretório pessoal.

Quando você digita um comando, ele aparecerá no cursor de texto. Você pode usar teclas como **home**, **end**, **backspace** e **setas** para editar o comando que você está digitando.

Depois de digitar o comando corretamente, pressione **Enter** para executá-lo.

4.2.1 Passo 1

O comando a seguir exibirá as mesmas informações exibidas na primeira parte do prompt. Certifique-se de ter selecionado (clique) na janela do **Terminal** primeiro e depois digite o seguinte comando seguido da tecla **Enter**:

```
whoami
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ whoami
sysadmin
sysadmin@localhost:~$
```

A saída do comando `whoami`, `sysadmin`, exibe o nome do usuário atual. Embora nesse caso seu nome de usuário seja exibido no prompt, esse comando pode ser usado para obter essas informações em uma situação em que o prompt não continha essas informações.

4.2.2 Passo 2

O próximo comando exibe informações sobre o sistema atual. Para poder ver o nome do kernel que você está usando, digite o seguinte comando no terminal:

```
uname
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ uname
Linux
```

Muitos comandos que são executados produzem uma saída de texto como essa. Você pode alterar qual saída é produzida por um comando usando opções após o nome do comando.

Opções para um comando podem ser especificadas de várias maneiras. Tradicionalmente no UNIX, as opções eram expressas por um hífen seguido por outro caractere, por exemplo: `-n`.

No Linux, às vezes, as opções também podem ser dadas por dois caracteres de hífen seguidos por uma palavra ou palavra hifenizada, por exemplo: `--nodename`.

Execute o comando `uname` novamente duas vezes no terminal, uma vez com a opção `-n` e novamente com a opção `--nodename`. Isso exibirá o nome do host do nó da rede, também encontrado no prompt.

```
uname -n
uname --nodename
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ uname -n
localhost
sysadmin@localhost:~$ uname --nodename
localhost
```

4.2.3 Passo 3

O comando `pwd` é usado para exibir seu "local" atual ou o diretório atual de trabalho. Digite o seguinte comando para exibir o diretório de trabalho:

```
pwd
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ pwd
/home/sysadmin
sysadmin@localhost:~$
```

O diretório atual no exemplo acima é `/home/sysadmin`. Isso também é chamado de seu diretório pessoal, um local especial onde você tem controle de arquivos e outros usuários normalmente não têm acesso. Por padrão, esse diretório é nomeado como seu nome de usuário e está localizado abaixo do diretório `/home`.

Como você pode ver na saída do comando, `/home/sysadmin`, o Linux usa a barra / como separador dos diretórios para criar o que é chamado de caminho (*path*). A barra inicial representa o diretório de nível superior, conhecido como o diretório *raiz*. Mais informações sobre arquivos, diretórios e caminhos serão apresentadas em laboratórios posteriores.

O caractere ~ til que você vê no seu prompt também está indicando qual é o diretório atual. Esse caracter é uma forma de "atalho" para representar seu diretório home.

Considere isto: `pwd` significa "imprimir diretório de trabalho". Embora na verdade não "imprima" em versões modernas, máquinas UNIX mais antigas não tinham monitores e a saída de comandos foi para uma impressora, daí o nome engraçado de `pwd`.

4.3 Variáveis Shell

As variáveis do shell são usadas para armazenar dados no Linux. Esses dados são usados pelo próprio shell, bem como por programas e usuários.

O foco desta seção é aprender como exibir os valores das variáveis do shell.

4.3.1 Passo 1

O comando `echo` pode ser usado para imprimir um texto, o valor de uma variável e mostrar como o ambiente de shell expande metacaracteres (mais sobre metacaracteres mais adiante neste laboratório). Digite o seguinte comando para que seja emitido o texto literal:

```
echo Hello Student
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo Hello Student
Hello Student
sysadmin@localhost:~$
```

4.3.2 Passo 2

Digite o seguinte comando para exibir o valor da variável `PATH`:

```
echo $PATH
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo $PATH

/home/sysadmin/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin:/usr/games

sysadmin@localhost:~$
```

A variável `PATH` é exibida colocando um caractere `$` na frente do nome da variável.

Esta variável é usada para encontrar a localização dos comandos. Cada um dos diretórios listados acima é pesquisado quando você executa um comando. Por exemplo, se você tentar executar o comando `date`, o shell procurará primeiro o comando no diretório `/home/sysadmin/bin` e, em seguida, no diretório `/usr/local/sbin` e assim por diante. Quando o comando `date` for encontrado, o shell "executa".

4.3.3 Passo 3

Use o comando `which` para determinar se existe um arquivo executável chamado `date`, localizado em um diretório listado no valor PATH:

Use the `which` command to determine if there is an executable file named `date` that is located in a directory listed in the PATH value:

```
which date
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ which date
/bin/date
sysadmin@localhost:~$
```

A saída do comando `which` informa que, quando você executar o comando `date`, o sistema executará o comando `/bin/date`. O comando `which` faz uso da variável `PATH` para determinar a localização do comando `date`.

4.4 Globbing

O uso de caracteres *glob* no Linux é semelhante ao que muitos sistemas operacionais chamam de caracteres "curinga". Usando caracteres glob, você combina nomes de arquivos usando padrões.

Caracteres Glob são um recurso de shell, não algo específico para qualquer comando específico. Como resultado, você pode usar caracteres glob com qualquer comando do Linux.

Quando caracteres glob são usados, o shell "expande" todo o padrão para corresponder a todos os arquivos no diretório especificado que corresponde ao padrão.

Para fins de demonstração, usaremos o comando `echo` para exibir este processo de expansão.

4.4.1 Passo 1

Use o seguinte comando `echo` para exibir todos os nomes de arquivos no diretório atual que corresponda ao padrão glob *:

```
echo *
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo *
Desktop Documents Downloads Music Pictures Public Templates Videos
sysadmin@localhost:~$
```

O asterisco `*` corresponde a "zero ou mais" caracteres em um nome de arquivo. No exemplo acima, isso resulta na correspondência de todos os nomes de arquivos no diretório atual.

O comando `echo`, por sua vez, exibe os nomes dos arquivos que foram correspondidos.

4.4.2 Passo 2

Os comandos a seguir exibirão todos os arquivos no diretório atual que começam com a letra `D` e a letra `P`:

```
echo D*
echo P*
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo D*
Desktop Documents Downloads
sysadmin@localhost:~$ echo P*
Pictures Public
sysadmin@localhost:~$
```

Pense no primeiro exemplo, `D*`, como "corresponde a todos os nomes de arquivos no diretório atual que começam com um caractere `d` maiúsculo e têm zero ou mais de qualquer outro caractere após o caractere `D` ".

4.4.3 Passo 3

O asterisco `*` pode ser usado em qualquer lugar da string. O comando a seguir exibirá todos os arquivos em seu diretório atual que terminam na letra `s`:

```
echo *s
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo *s
Documents Downloads Pictures Templates Videos
sysadmin@localhost:~$
```

4.4.4 Passo 4

Observe que o asterisco também pode aparecer várias vezes ou no meio de vários caracteres:

```
echo D*n*s
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo D*n*s
Documents Downloads
sysadmin@localhost:~$
```

O próximo metacaractere glob que iremos examinar é o ponto de interrogação `?`. O ponto de interrogação corresponde exatamente a um caractere. Esse caractere único pode ser qualquer caractere possível.

Como o asterisco, ele pode ser usado em qualquer lugar de uma string e pode aparecer várias vezes.

4.4.5 Passo 5

Como cada ponto de interrogação corresponde a um caractere desconhecido, digitar seis deles corresponderá a seis caracteres de nomes de arquivo. Digite o seguinte para exibir os nomes dos arquivos com exatamente seis caracteres:

```
echo ??????
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo ??????
Public Videos
sysadmin@localhost:~$
```

Importante: cada caractere `?` deve corresponder exatamente a um caractere em um nome de arquivo, não mais e não menos que um caractere.

4.4.6 Passo 6

Usar o ponto de interrogação com outros caracteres limitará as correspondências. Digite o seguinte para exibir os nomes dos arquivos que começam com a letra D e têm exatamente nove caracteres:

```
echo D????????
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo D????????
Documents Downloads
sysadmin@localhost:~$
```

.4.7 Passo 7

Caracteres curinga ou glob podem ser combinados. O comando a seguir exibirá nomes de arquivo com pelo menos seis caracteres e terminará com a letra s.

```
echo ?????*s
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo ?????*s
Documents Downloads Pictures Templates Videos
sysadmin@localhost:~$
```

Pense no padrão `?????*s` para significar "nomes de arquivos de correspondência que começam com cinco caracteres, então, possuem zero ou mais de quaisquer caracteres e, em seguida, terminam com um caractere `s` ".

4.4.8 Passo 8

O próximo glob é semelhante ao glob ponto de interrogação para especificar um caractere.

Essa glob usa um par de colchetes `[ ]` para especificar qual caractere será permitido. Os caracteres permitidos podem ser especificados como um intervalo, uma lista ou o que é conhecido como classe de caracteres.

Os caracteres permitidos também podem ser negados com um ponto de exclamação `!`.

No primeiro exemplo, o primeiro caractere do nome do arquivo pode ser um `D` ou um `P`. No segundo exemplo, o primeiro caractere pode ser qualquer caractere, exceto um `D` ou `P`:

```
echo [DP]*
echo [!DP]*
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo [DP]*
Desktop Documents Downloads Pictures Public
sysadmin@localhost:~$ echo [!DP]*
Music Templates Videos
sysadmin@localhost:~$
```

4.4.9 Passo 9

Nos próximos exemplos, um intervalo de caracteres será especificado. No primeiro exemplo, o primeiro caractere do nome do arquivo pode ser qualquer caractere começando em `D` e terminando em `P`. No segundo exemplo, esse intervalo de caracteres é negado, o que significa que qualquer caractere único corresponderá, desde que não esteja entre os caracteres. letras `D` e `P`:

```
echo [D-P]*
echo [!D-P]*
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo [D-P]*
Desktop Documents Downloads Music Pictures Public
sysadmin@localhost:~$ echo [!D-P]*
Templates Videos
sysadmin@localhost:~$
```

Você pode estar se perguntando "quem decide quais letras vêm entre `D` e `P` ?". Nesse caso, a resposta é bastante óbvia (E, F, G, H, I, J, K, L, M, N e O), mas e se o intervalo fosse `[1-A]`?

A tabela de texto ASCII é usada para determinar o intervalo de caracteres. Você pode visualizar essa tabela procurando por ela na Internet ou digitando o seguinte comando: `ascii`

Então, quais caracteres o glob `[1-A]` associam? De acordo com a tabela de texto ASCII: `1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `:`, `;`, `<`, `=`, `>`, `?`, `@` e `A`.

4.5 Aspas

Existem três tipos de citações usadas pelo shell Bash: aspas simples (`'`), aspas duplas (`"`) e as aspas invertidas ou crase (`` ` ``). Essas aspas têm características especiais no shell Bash, conforme descrito abaixo.

Para entender aspas simples e duplas, considere que há momentos em que você não quer que o shell trate alguns caracteres como "especiais". Por exemplo, como você aprendeu anteriormente neste laboratório, o caractere `*` é usado como curinga. E se você quisesse que o caractere `*` significasse apenas um asterisco?

Aspas simples impedem o shell de "interpretar" ou expandir todos os caracteres especiais. Muitas vezes, aspas simples são usadas para proteger uma string de ser alterada pelo shell, para que a string possa ser interpretada por um comando como um parâmetro para afetar a maneira como o comando é executado.

Aspas duplas param a expansão de caracteres glob como o asterisco (`*`), ponto de interrogação (`?`) E colchetes (`[ ]`). Aspas duplas permitem que a expansão de variáveis e a substituição de comandos ocorram.

As aspas invertidas ou crase causam "substituição de comando" que permite que um comando seja executado dentro da linha de outro comando.

Ao usar aspas, elas devem ser inseridas em pares ou então o shell não considerará o comando completo.

Enquanto aspas simples são úteis para bloquear o shell de interpretar um ou mais caracteres, o shell também fornece uma maneira de bloquear a interpretação de apenas um único caractere chamado "escape" do caractere. Para "escapar" o significado especial de um metacaractere de shell, o caractere de barra invertida `\` é usado como um prefixo para esse caractere.

4.5.1 Passo 1

Execute o seguinte comando para usar aspas invertidas (crase) ` para executar o comando `date` dentro da linha do comando `echo`:

```
echo Today is `date`
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo Today is `date`
Today is Tue Jan 19 15:48:57 UTC 2016
sysadmin@localhost:~$
```

4.5.2 Passo 2

Você também pode colocar `$` (antes do comando e) após o comando para realizar a substituição do comando:

```
echo Today is $(date)
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo Today is $(date)
Today is Tue Jan 19 15:51:09 UTC 2016
sysadmin@localhost:~$
```

Por que dois métodos diferentes que realizam a mesma coisa? Aspas invertidas são muito semelhantes às aspas simples, tornando mais difícil "ver" o que um comando deve fazer. Originalmente os shells usavam aspas invertidas; O formato `$(command)` foi adicionado em uma versão posterior do shell Bash para tornar a instrução mais clara visualmente.

4.5.3 Passo 3

Se você não quiser que as aspas invertidas sejam usados para executar um comando, coloque aspas simples ao redor deles. Execute o seguinte:

```
echo This is the command '`date`'
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo This is the command '`date`'
This is the command `date`
sysadmin@localhost:~$
```

4.5.4 Passo 4

Observe que você também pode colocar um caractere de barra invertida na frente de cada caractere de aspa invertida. Execute o seguinte:

```
echo This is the command \`date\`
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo This is the command \`date\`
This is the command `date`
sysadmin@localhost:~$
```

4.5.5 Passo 5

Caracteres de aspas duplas não afetam os caracteres aspas invertidas. O shell ainda os usará como substituição de comando. Execute o seguinte para ver uma demonstração:

```
echo This is the command "`date`"
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo This is the command "`date`"
This is the command Tue Jan 19 16:05:41 UTC 2016
sysadmin@localhost:~$
```

4.5.6 Passo 6

Caracteres de aspas duplas afetarão os caracteres curinga, desativando seu significado especial. Execute o seguinte:

```
echo D*

echo "D*"
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo D*
Desktop Documents Downloads
sysadmin@localhost:~$ echo "D*"
D*
sysadmin@localhost:~$
```

Importante:

As citações podem parecer triviais e estranhas no momento, mas à medida que você ganha mais experiência trabalhando no shell de comando, descobrirá que ter um bom entendimento de como as cotações diferentes funcionam é fundamental para usar o shell.

4.6 Instruções de Controle

Normalmente, você digita um único comando e o executa quando pressiona **Enter**. O shell Bash oferece três instruções diferentes que podem ser usadas para separar vários comandos digitados.

O separador mais simples é o ponto-e-vírgula (`;`). Usar o ponto e vírgula entre vários comandos permite que sejam executados um após o outro, sequencialmente da esquerda para a direita.

Os caracteres `&&` criam uma lógica e uma declaração. Comandos separados por `&&` são condicionalmente executados. Se o comando à esquerda do `&&` for bem sucedido, então o comando à direita do `&&` também será executado. Se o comando à esquerda do `&&` falhar, o comando à direita do `&&` não será executado.

O `||` caracteres criam uma lógica ou declaração, que também causa execução condicional. Quando os comandos são separados por `||`, então somente se o comando à esquerda falhar, o comando fica à direita do `||` executar. Se o comando à esquerda do `||` sucede, então o comando à direita do `||` não será executado.

Para ver como essas instruções de controle funcionam, você estará usando dois executáveis especiais: `true` e `false`. O executável `true` sempre tem sucesso quando é executado, enquanto o executável `false` sempre falha. Embora isso possa não fornecer exemplos realistas de como `&&` and `||` trabalham, ele fornece um meio para demonstrar como eles funcionam sem ter que introduzir novos comandos.

4.6.1 Passo 1

Execute os seguintes três comandos juntos separados por ponto e vírgula:

```
echo Hello; echo Linux; echo Student
```

Como você pode ver, a saída mostra todos os três comandos executados sequencialmente:

```
sysadmin@localhost:~$ echo Hello; echo Linux; echo Student
Hello
Linux
Student
sysadmin@localhost:~$
```

4.6.2 Passo 2

Agora, coloque três comandos juntos separados por ponto e vírgula, onde o primeiro comando é executado com um resultado de falha:

```
false; echo Not; echo Conditional
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ false; echo Not; echo Conditional
Not
Conditional
sysadmin@localhost:~$
```

Observe que, no exemplo anterior, todos os três comandos ainda foram executados, embora o primeiro tenha falhado. Enquanto você não pode ver a saída do comando `false`, ele foi executado. No entanto, quando os comandos são separados pelo caractere `;`, eles são completamente independentes um do outro.

4.6.3 Passo 3

Em seguida, use o "e lógico " para separar os comandos:

```
echo Start && echo Going && echo Gone
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo Start && echo Going && echo Gone
Start
Going
Gone
sysadmin@localhost:~$
```

Como cada instrução `echo` é executada corretamente, um valor de retorno de sucesso é fornecido, permitindo que a próxima instrução também seja executada.

4.6.4 Passo 4

Use o "e lógico " com um comando que falha como mostrado abaixo:

```
echo Success && false && echo Bye
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo Success && false && echo Bye
Success
sysadmin@localhost:~$
```

O primeiro comando `echo` é bem-sucedido e vemos sua saída. O comando `false` é executado com um resultado de falha, portanto, a última instrução `echo` não é executada.

4.6.5 Passo 5

Os caracteres "ou" que separam os comandos a seguir demonstram como a falha antes da instrução "ou" faz com que próximo comando seja executado; no entanto, uma primeira instrução bem-sucedida faz com que o comando não seja executado:

```
false || echo Fail Or

true || echo Nothing to see here
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ false || echo Fail Or
Fail Or
sysadmin@localhost:~$ true || echo Nothing to see here
sysadmin@localhost:~$
```

4.7 Histórico do Shell

O shell Bash mantém um histórico dos comandos que você digita. Comandos anteriores podem ser facilmente acessados neste histórico de várias maneiras.

A primeira e mais fácil maneira de recuperar um comando anterior é usar a **tecla de seta para cima**. Cada vez que você pressionar a **tecla de seta para cima**, voltará um comando pelo histórico. Se você acidentalmente voltar muito longe, a **tecla de seta para baixo** avançará no histórico de comandos.

Quando você encontrar o comando que deseja executar, poderá usar as **teclas de seta para a esquerda** e as **teclas de seta para a direita** para posicionar o cursor para edição. Outras teclas úteis para edição incluem as teclas **Home**, **End**, **Backspace** e **Delete**.

Outra maneira de usar seu histórico de comandos é executar o comando `history` para poder visualizar uma lista de histórico numerado. O número listado à esquerda do comando pode ser usado para executar o comando novamente. O comando `history` também possui um número de opções e argumentos que podem manipular quais comandos serão armazenados ou exibidos.

4.7.1 Passo 1

Execute alguns comandos e execute o comando `history`:

```
date
clear
echo Hi
history
```

Lembre-se: O comando `date` imprimirá a hora e a data no sistema. O comando `clear` limpa a tela.

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ echo Hi
Hi
sysadmin@localhost:~$ history
    1 date
    2 clear
    3 echo Hi
    4 history
sysadmin@localhost:~$
```

Seus números de comando provavelmente serão diferentes daqueles fornecidos acima. Isso porque você provavelmente executou um número diferente de comandos.

4.7.2 Passo 2

Para exibir um número limitado de comandos, o comando `history` pode usar um número como parâmetro para exibir exatamente essas entradas recentes. Digite o seguinte comando para exibir os últimos cinco comandos do seu histórico:

```
history 5
```

Sua saída deve ser semelhante à seguinte:

```
sysadmin@localhost:~$ history 5
   185  false || Fail Or
   186  false || echo Fail Or
   187  true || echo Nothing to see here
   188  history
   189  history 5
sysadmin@localhost:~$
```

4.7.3 Passo 3

Para executar um comando novamente, digite o ponto de exclamação e o número da lista de histórico. Por exemplo, para executar o 94º comando na sua lista de histórico, você executaria o seguinte:

```
!94
```

4.7.4 Passo 4

Em seguida, experimente acessar seu histórico usando as **teclas de seta para cima** e as **teclas de seta para baixo**. Continue pressionando a **tecla de seta para cima** até encontrar um comando que você deseja executar. Se necessário, use outras teclas para editar o comando e pressione **Enter** para executar o comando.