Notas de Aula gentilmente disponibilizadas pelo aluno Pedro Henrique

Aula 103.1 – Execução de Comandos em Sequência

O que veremos nesta aula:

Dando Continuidade à aula anterior, vamos ver mais algumas propriedades do bash e alguns outros comandos.

Sequência de Comandos

Habitualmente no Linux alguns usuários geralmente digitam comando por comando, porém é possível fazer com que o sistema execute os comandos sequencialmente, e há várias formas de fazer isso, vejamos:

O separador ";"

A primeira forma que é explicada na aula é o separador "**;**", através deles podemos executar um comando atrás do outro, ele tem uma particularidade que é, independente se o comando está certo ou errado ele é executado.

```
# clear ; date ; ls
```

Sendo assim, ele executa um comando por vez não importando se o comando anterior, no caso clear esteja certo ou não, ele executará em os demais comandos.

O comando && (E comercial)

Diferente do "**;**" o **&&** acusa erro para de executar se algum dos comandos estiverem errados.
Por exemplo: Se o primeiro comando for executado com sucesso ele parte para o segundo comando, caso contrário ele para a execução acusando o erro.

```
# ls /tmp/teste && echo Linux
```

Note que **primeiro** executamos o ls apontando o arquivo /teste dentro do diretório /tmp e **em seguida** ele deve executar o comando echo, caso não exista o arquivo dentro do diretório /tmp ele acusa erro e para a execução.

```
lpi1@linux:~/Desktop$ ls /tmp/teste && echo Linux
ls: cannot access '/tmp/teste': No such file or directory
lpi1@linux:~/Desktop$
```

IMPORTANTE: O comando && entende que:

Faça isso **E** isso **= faça ls && echo**

Quando o arquivo existe ele apenas executa normalmente.

```
lpi1@linux:~/Desktop$ ls /tmp/teste && echo Linux
/tmp/teste
Linux
lpi1@linux:~/Desktop$
```

O Comando || (Pipe)

O separador pipe faz o inverso do &&, ele executa o segundo comando caso o primeiro comando falhe. Vejam:

```
lpi1@linux:~/Desktop$ ls /tmp/curso_lpi1 || echo Linux
ls: cannot access '/tmp/curso_lpi1': No such file or directory
Linux
lpi1@linux:~/Desktop$
```

Se o primeiro comando der certo ele ignora a regra do pipe e apenas executa o primeiro comando existente.

IMPORTANTE: O comando || entende:

Faça isso **OU** isso = **faça ls || echo**

```
lpi1@linux:~/Desktop$ ls /tmp/curso_lpi1 || echo Linux
/tmp/curso_lpi1
lpi1@linux:~/Desktop$
```

Sendo assim, qualquer um dos comandos que tenham sucesso impostas pelo || ele encerra a execução.

Veja que nesse exemplo abaixo, ele acusa erro no primeiro comando, executa o seguindo comando e logo para, ignorando assim o terceiro comando.

```
lpi1@linux:~/Desktop$ ls /tmp/curso_ || echo Linux || cd /usr/local/
ls: cannot access '/tmp/curso_': No such file or directory
Linux
lpi1@linux:~/Desktop$
```

O comando !!

O comando **!!** repete o último comando executado no bash.

!!

```
lpi1@linux:/home$ ls /tmp/
config-err-kgmMQj
systemd-private-14aa890156d746ffb7fdf27c94d029bb-rtkit-daemon.service-5Ux0O2
systemd-private-14aa890156d746ffb7fdf27c94d029bb-systemd-timesyncd.service-0sHT7
L
lpi1@linux:/home$ !!
ls /tmp/
config-err-kgmMQj
systemd-private-14aa890156d746ffb7fdf27c94d029bb-rtkit-daemon.service-5Ux0O2
systemd-private-14aa890156d746ffb7fdf27c94d029bb-systemd-timesyncd.service-0sHT7
L
lpi1@linux:/home$
```

Repetição de comandos

O comando history

O comando **history** lista os todos os últimos comandos digitados no bash, cada usuário tem o seu arquivo de histórico.

history

```
    1  echo $CURSOLINUX
    2  exit
    3  echo $PATH
    4  echo PATH
    5  NOME_VARIAVEL=valor
    6  echo $NOME_VARIAVEL
    7  CURSOLINUX=lpi1
    8  echo $CURSOLINUX
    9  env
   10  set
   11  /home/lpi1/
   12  cd /home/lpi1/
   13  bash
   14  echo $CURSOLINUX
   15  bash
   16  exsit
   17  exit
   18  poweroff
   19  echo $CURSOLINUX
   20  echo $NOME_VARIAVEL
   21  NOME_VARIAVEL=valor
   22  CURSOLINUX=lpi1
   23  bash
   24  ls
   25  clear
   26  uname
   27  man uname
   28  ls -a
   29  ls -la
   30  pwd
   31  history
   32  cd /home/
   33  ls -l
   34  ls -l /tmp/
   35  ls /tmp/
   36  history
lpi1@linux:/home$
```

Como vemos na imagem acima, o comando lista os últimos comando digitados, e na antes de cada comando há uma número.

Outra forma que podemos executar novamente o comando é da seguinte forma:

```
# !26
```

Sendo assim ele repetira o comando 26, conforme a lista acima.

Uma outra forma ainda, é o "!" seguido da string, por exemplo:

```
# !uname
```

Dessa forma ele buscará o comando e o executará da forma que foi executado pela última vez.

Outro exemplo é

```
# !ls
```

Como limpar o arquivo de Histórico

Para limparmos o arquivo de histórico do nosso usuário executamos o comando abaixo:

```
# history -c
```

```
lpi1@linux:/home$ history -c
lpi1@linux:/home$ history
    1  history
lpi1@linux:/home$
```

Os comandos digitados ficam armazenados no arquivo **.bash_history**, este arquivo é encontrado dentro da pasta do usuário em /home/lpi1 como no exemplo:

```
lpi1@linux:~$ pwd
/home/lpi1
lpi1@linux:~$ cat .bash_history
echo $CURSOLINUX
exit
echo $PATH
echo PATH
NOME_VARIAVEL=valor
echo $NOME_VARIAVEL
CURSOLINUX=lpi1
echo $CURSOLINUX
env
set
/home/lpi1/
cd /home/lpi1/
bash
echo $CURSOLINUX
bash
exsit
exit
poweroff
echo $CURSOLINUX
echo $NOME_VARIAVEL
NOME_VARIAVEL=valor
CURSOLINUX=lpi1
bash
lpi1@linux:~$
```

Mas, não havíamos zerado o arquivo history?

Quando fazemos o **login**, o que está no .bash_history é carregado em memória, e o comando history funciona com o que está na memória, quando fazemos faz o **logout** é feito um append do que há de novo na memória para o arquivo .bash_history.

Quando usamos o history -c, os comandos do history em memória é limpo, e quando você faz o logout, é feito um append de nada no .bash_history, ou seja, ele não é alterado.

Para limpar definitivamente há 2 opções, executar os comandos abaixo, forçando que o history em memória, em branco, seja refletido no .bash_history:

```
# history -c && history -w
```

Ou simplesmente limpar manualmente o arquivo:

```
# history -c
```

```
# cat /dev/null > ~/.bash_history
```

Pesquisando comandos digitados

No bash ainda temos a possibilidade de buscar comandos que já digitamos anteriormente, para abrira caixa de pesquisa pressionamos o **ctrl+R.** Este procura os comandos dentro do seu histórico de comandos.

Conforme digitamos o comando desejado ele já mostra os comandos iniciados com aquela string. Encontrando o comando, basta teclar o enter.

```
(reverse-i-search)`ls': ls -l
```

O Auto Completar

Essa função se dá quando digitamos um comando seja ele para arquivo ou diretórios ele completa o comando quando pressionamos a tecla tab. O auto completar tem suas particularidades, ver aula 10 para verificar os exemplos.

Comandos de ajuda

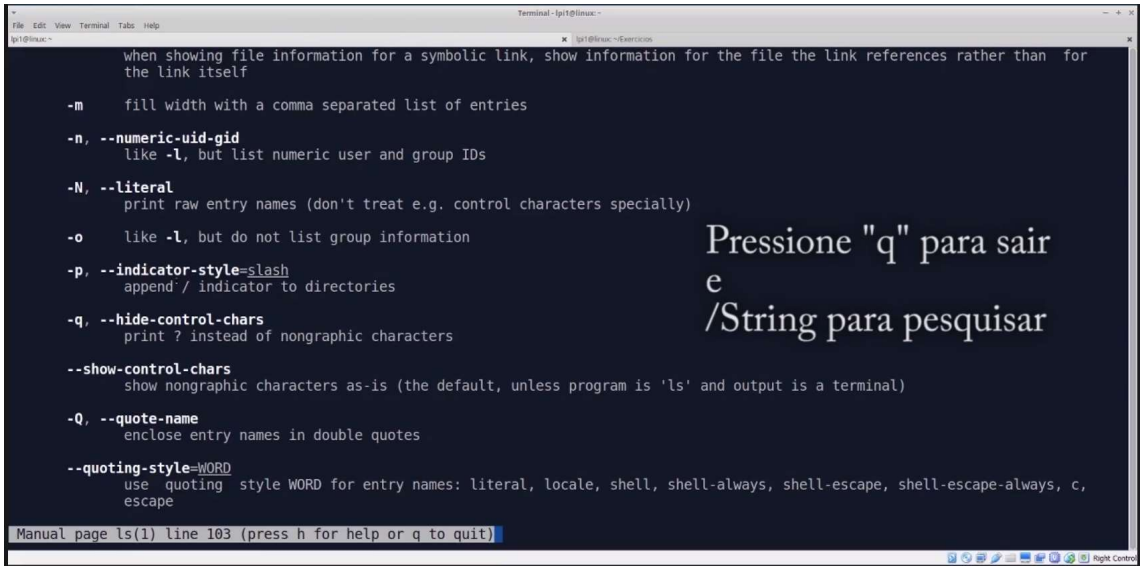
Um dos recurso muito importante do shell são os comandos para obter ajuda.

O comando man

O comando **man** mostra o manual de ajuda do comando, basicamente todos os comandos tem o seu manual de referência, e eles são acessados pelo **man** exemplo:

```
# man ls
```

Após o comando ser executado é aberto o manual de referência completo do comando **ls**.



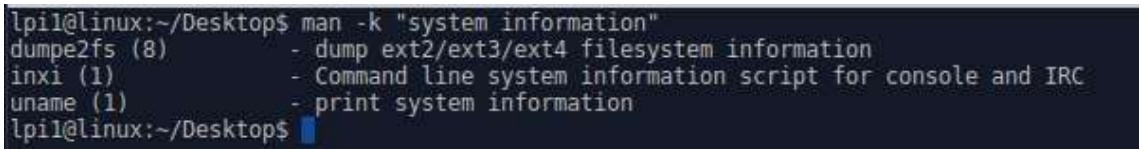
Uma observação é que, quando o comando é interno, ou seja, faz parte do bash ele não possui o man, sendo assim temos que consultar o manual do bash:

```
# man bash
```

Outra forma de usar o man é com o parâmetro **–k**:

```
# man –k “system infomation”
```

Dessa forma o comando traz qualquer referência que contenha o conteúdo “system information” e o comando do conteúdo informado. Veja:



Novamente:

```
# man –k “update system”
```

O comado info

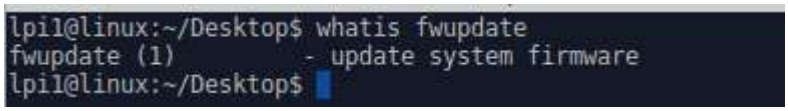
Um pouco diferente do **man** é o comando **info**, ele basicamente é um man de forma reduzida, o tanto o **man** quanto o **inf** são comandos para buscar ajuda sobre determinados comandos:

```
# Info ls
```

O comando whatis e apropos

O comando whatis consulta a descrição do comando, assim como o man –k:

```
# whatis fwupdate
```



Já o comando **apropos** faz a busca baseado na descrição do comando e traz o comando referente a ela:

```
# apropos “update system”
```