Linha de comando. Shell e comandos

Agosto/2017

Prof. Jairo

jairo@uni9.pro.br

professor@jairo.pro.br

http://www.jairo.pro.br/

Este material tem por única intenção reunir um conteúdo acadêmico necessário para auxiliar no ensino da disciplina "Prática e Administração de Sistemas Operacionais de Redes Livres", ministrado nos cursos de Tecnologias em Redes de Computadores e Segurança da Informação.

O conteúdo aqui exposto pode ser livremente redistribuído e usado como apoio de aula, desde que mantenha a sua integridade original.

O arquivo "comandos-shell.pdf" pode ser livremente acessado em "http://www.jairo.pro.br/prat_adm_sist_oper/".

Qualquer crítica ou sugestão, favor entrar em contato com o Prof. Jairo no endereço eletrônico "jairo@uni9.pro.br" ou "professor@jairo.pro.br".

Sumário:

1 - LINHA DE COMANDO

A única interface comum a todos os sistemas membros da família Unix é a linha de comando, portanto toda a administração desses sistemas é feita a partir de comandos.

Os comandos são disparados de uma aplicação chamada shell. O shell também é conhecido por interpretador de comandos. Ao contrário do DOS/Windows, no mundo Unix existem dezenas de interpretadores de comandos, por exemplo:

sh: Bourne Shell
bash: Bourne Shell Again
ksh: Korn Shell
csh: C shell

Para a linha de comando, basicamente todo shell é igual. Porém, para escrever scripts [shell script] precisa usar a linguagem específica de cada shell.

2 - CONCEITOS BÁSICOS DE COMANDOS

Todo comando envolve a execução de pelo menos um executável. Por exemplo, ao ser comandado ***ls***, o usuário está disparando o executável /bin/ls, que gera um processo.

De um modo geral, todo comando admite que seja passado pelo menos um parâmetro ou opção, além de arquivo.

Por exemplo, o comando ***ls -l /etc*** lista o conteúdo do diretório ***/etc*** no formato longo. Foi passado o parâmetro ***-l*** e o arquivo ***/etc***. Se não tivesse sido passado parâmetro nem arquivo, o comando seguiria o seu *default* [padrão], que no caso do ***ls*** é listar o diretório atual.

De forma análoga ao ***ls***, a maioria dos comandos admite receber parâmetros e arquivos.

3 - EXEMPLOS DE COMANDOS

A notação usada nestes exemplos é:

```
o comando:              shell$ ls
a saída do comando:     arquivo1    dir3
```

Para saber em qual diretório do sistema hierárquico de arquivo o usuário se encontra, usar o comando **pwd** (**p**rint **w**orking **d**irectory):

```
shell$ pwd
/home/aluno
```

Para se deslocar através do sistema hierárquico de arquivo, usar o comando **cd**:

```
shell$ cd  /etc
shell$ pwd
/etc
```

Listagem do diretório /bin:

```
shell$ ls  /bin
alsaunmute      dbus-monitor    false       link        nice          rvi         tracepath6
arch            dbus-send       fgrep       ln          nisdomainname rview       traceroute
awk             dbus-uuidgen    find        loadkeys    ntfs-3g       sed         traceroute6
basename        dd              fusermount  login       ntfs-3g       probe       setfont
....
```

Para listar no formato longo, usar a opção ***-l***:

```
shell$ ls  -l  /bin
-rwxr-xr-x.  1  root  root      123 Mai  15 09:38 alsaunmute
-rwxr-xr-x.  1  root  root    36608 Abr   1 08:20 arch
lrwxrwxrwx. 1  root  root        4 Jun  14 15:19 awk -> gawk
-rwxr-xr-x.  1  root  root    34720 Abr   1 08:20 basename
-rwxr-xr-x.  1  root  root  838824 Abr   8 07:46 bash
-rwxr-xr-x.  1  root  root    57096 Abr   1 08:20 cat
-rwxr-xr-x.  1  root  root    63804 Abr   1 08:20 chgrp
-rwxr-xr-x.  1  root  root    59652 Abr   1 08:20 chmod
-rwxr-xr-x.  1  root  root    66608 Abr   1 08:20 chown
-rwxr-xr-x.  1  root  root  110996 Abr   1 08:20 cp
....
```

NOTA: todo nome de arquivo precedido de uma barra direita (por exemplo, **/etc**) representa um caminho (path) **absoluto**. Se não for precedido dessa barra, é um caminho **relativo**. Por exemplo, se o usuário está no diretório /bin e comanda "***ls -l mkdir***", o caminho é relativo; se comandasse "***ls -l /bin/mkdir***", o caminho seria absoluto.

Para determinar o tipo do arquivo, usar o comando **file**:

```
shell$ file /etc/group
/etc/group: ASCII text

shell$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked (uses shared libs),
for GNU/Linux 2.6.18, stripped

shell$ file /usr
/usr: directory
```

Para visualizar o conteúdo de um **arquivo de texto**, usar o comando more:

```
shell$ more /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
vcsa:x:69:69:virtual console memory owner:/dev:/sbin/nologin
avahi-autoipd:x:499:499:avahi-autoipd:/var/lib/avahi-autoipd:/sbin/nologin
--Mais--(41%)
```

NOTA: para saber de que tipo é o arquivo, usar o comando **file**.

No arquivo ***/etc/passwd*** é onde estão cadastrados os usuários locais ao sistema.

No arquivo ***/etc/group*** estão cadastrados os grupos locais ao sistema.

Num Linux, no arquivo ***/etc/shadow*** é onde estão cadastrada as senhas de usuários locais.

O comando **cat** é usado para pegar e mostrar todo o conteúdo de um **arquivo de texto** de uma única vez:

```
shell$ cat /etc/inittab
# Default runlevel. The runlevels used are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:
```

NOTA: o comando **tac** é semelhante ao cat, porém apresenta a saída do arquivo na ordem inversa: a última linha é mostrada como a primeira, a penúltima como a segunda, e assim por diante.

O comando **grep** busca expressão regular (ou palavra) no conteúdo do arquivo:

```
shell$ grep operator /etc/passwd
operator:x:11:0:operator:/root:/sbin/nologin
```

Com a opção **-v**, **grep** faz uma busca reversa, isso para ignorar todas as linhas que contenham o padrão de busca:

```
shell$ grep -v "#" /etc/inittab
id:5:initdefault:
```

Com a opção **-n**, **grep** mostra o(s) número(s) da(s) linha(s) aonde foi encontrada expressão regular ou palavra:

```
shell$ grep -n "/bin/bash" /etc/passwd
1:root:x:0:0:root:/root:/bin/bash
46:aluno:x:1067:1067::/home/aluno:/bin/bash
```

Para visualizar o conteúdo do arquivo de texto com numeração de linhas, usar o comando **nl**:

```
shell$ nl  -n  ln  /etc/passwd
1       root:x:0:0:root:/root:/bin/bash
2       bin:x:1:1:bin:/bin:/sbin/nologin
3       daemon:x:2:2:daemon:/sbin:/sbin/nologin
4       adm:x:3:4:adm:/var/adm:/sbin/nologin
...
```

No comando acima, a opção "***-n nl***" é para apresentar a numeração de linhas no formato com justificação à esquerda.

Para visualizar as últimas linhas de um **arquivo de texto**, usar o comando **tail**:

```
shell$ tail  /etc/group
stapdev:x:491:
stapusr:x:490:
wbpriv:x:88:squid
smolt:x:489:
torrent:x:488:
haldaemon:x:68:
squid:x:23:
gdm:x:42:
aluno:x:500:
jackuser:x:487:
```

NOTA: o *default* do tail é mostrar as 10 últimas linhas. Se fosse para mostrar apenas as 4 últimas linhas, bastaria comandar:

```
shell$ tail  -4  /etc/group
squid:x:23:
gdm:x:42:
aluno:x:500:
jackuser:x:487:
```

NOTA: **tail** com opção **-f** mostra as últimas linhas de um arquivo e permanece tentando ler novas linhas, à medida que forem sendo escritas. Isso é muito útil para acompanhar acessos em arquivos de logs. Por exemplo: **tail -f /var/log/messages**

Para visualizar as linhas do início de um arquivo de texto, usar o comando **head**:

```
shell$ head  /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

NOTA: o *default* do head é mostrar as 10 primeiras linhas. Do mesmo jeito que o comando tail, é só passar como opção o número de linhas que quiser visualizar.

Para contar o número de linhas, palavras e caracteres de um arquivo de texto, usar o comando **wc** (**w**ord **c**ount):

```
shell$ wc  /etc/passwd
40   65  1986  /etc/passwd
```

NOTA: o arquivo /etc/passwd, acima, tem 40 linhas, 65 palavras e 1986 caracteres.

Para copiar um arquivo, usar o comando **cp**:

```
shell$ cp  /etc/passwd  /tmp
```

NOTA: para copiar um arquivo precisa informar tanto a origem quanto o destino desta cópia. Precisa também permissão de leitura no arquivo de origem e permissão de escrita no diretório de destino.

Para comparar dois arquivos, usar o comando **cmp**:

```
shell$ cmp  /etc/passwd  /tmp/passwd
```

NOTA: a comparação do cmp é byte à byte.

Para mover um arquivo, usar o comando **mv**:

```
shell$ mv  /tmp/passwd  /tmp/passwd2
```

NOTA: para mover, além dos pré-requisitos do comando cp, precisa também ser dono do arquivo de origem, isso para poder excluí-lo no final.

Para criar um diretório, usar o comando **mkdir** (**ma**k**e dir**ectories):

```
shell$ mkdir  /home/aluno/teste
```

Para remover um diretório vazio, usar o comando **rmdir** (**re**m**ove dir**ectories):

```
shell$ rmdir  /home/aluno/teste
```

Para remover (excluir) um arquivo regular, usar o comando **rm**:

```
shell$ rm  /tmp/passwd2
```

NOTA: para remover um diretório com conteúdo, usar o comando **rm** com opção **-r** [recursivo]. Por exemplo:

```
shell$ rm  -r  /home/aluno/teste
```

Para alterar a data de um arquivo, usar o comando **touch**:

```
shell$ touch  teste.txt
```

NOTA: caso não exista, o comando touch cria um arquivo de texto vazio (sem conteúdo).

Para pedir o menu de ajuda [help] do comando ls, usar a opção **--help**:

```
shell$ ls  --help
```

NOTA: não existe padronização em torno do "--help". Em muitos comandos, a ajuda vem

com a opção "-h".

Para invocar o manual de ajuda de um comando, usar o comando **man** e passar como opção o nome do comando para o qual quer obter o manual:

```
shell$ man  ls
```

NOTA: nem sempre o man está instalado.

Para ordenar a apresentação do conteúdo de um arquivo, usar o comando **sort**:

```
shell$ sort  /etc/passwd
```

Para apresentar recortes de campos de um arquivo, usar o comando **cut**:

```
shell$ cut  -d:  -f1  /etc/passwd
```

NOTA: para o comando cut funcionar, o arquivo precisa ter uma estrutura de dados, isto é, precisa que os dados estejam distribuídos em colunas, e que todas as linhas tenham o mesmo número de colunas.

Para encontrar todos os arquivos de nome *passwd* no diretório /etc, usar o comando **find**:

```
shell$ find  /etc  -type  f  -name  passwd  -print
```

NOTA 1: as opções do comando find, acima, são:

-type f => encontra apenas arquivos regulares. Para buscar diretório, usar "d";

-name passwd => para que o arquivo tenha exatamente o nome "passwd". Poderia usar curingas, por exempo "*asswd*", que encontraria tanto "Passwd" quanto "old-passwd2".

NOTA 2: para tornar a busca insensível à caixa, usar a opção "-iname" ao invés de "-name".

O comando **find** também pode ser usado para encontrar arquivos que contêm determinado conteúdo. Por exemplo, para encontrar todos os arquivos que contenham a palavra aluno no diretório /etc, comandar:

```
shell$ find /etc -type f -exec grep -i aluno {} \; -print
```

O comando **sed** (**s**tream **ed**itor), possibilita (entre outras coisas) fazer filtragem e substituição. Por exemplo, no arquivo cores.txt, abaixo, o comando **sed** é usado para alterar todas as ocorrências da palavra vermelho para branco:

```
shell$ cat cores.txt
céu azul
grama verde
carro vermelho
vermelho claro
```

```
shell$ sed 's/vermelho/branco/g' cores.txt
céu azul
grama verde
carro branco
branco claro
```

Ao invés do comando sed, o **awk** também poderia ser usado para encontrar e substituir:

```
shell$ awk '{sub("vermelho", "branco", $2);}1' cores.txt
céu azul
grama verde
carro branco
vermelho claro
```

NOTA: na substituição acima, o awk somente considerou a cor vermelho encontrada no segundo campo.

Para visualizar o valor de alguma variável (por exemplo, do ambiente), pode ser usado o comando **echo**:

```
shell$ echo $SHELL
```

```
shell$ echo $HOME
```

NOTA 1: SHELL e HOME são as variáveis de ambiente para o shell e para a home do usuário. Mas para mostrar o valor que contém, a variável precisa ser precedida do cifrão "$". E como o nome destas variáveis foram definidas em caixa alta (letras maiúsculas), precisam ser chamadas exatamente assim no comando.

NOTA 2: o comando **env** (environment) é para visualizar todas as variáveis do ambiente.

Para ganhar privilégios de root, usar o comando **sudo**:

```
shell$ sudo su -
[sudo] password for aluno:
root# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

NOTA: o comando **id** apenas mostra a identificação do usuário.

Para retornar ao shell de aluno, comandar **exit**:

```
root# exit
shell$
```

4 - PIPES

Os pipes são implementação de comunicação entre processos. Na linha de comando, se for disparado mais de um executável de uma vez, vai gerar mais de um processo. E se esses executáveis estiverem ligados por um pipe "|", então a saída de um processo será a entrada do outro processo. Esta sequência de comunicação vai da esquerda para a direita.

Por exemplo:

```
shell$ cat /etc/passwd | more
```

Acima, **cat /etc/passwd** captura numa lista o conteúdo do arquivo /etc/passwd, porém ao invés de enviar para o monitor (shell), envia para o pipe (que é um buffer[1]). Do lado direito do pipe, **more** lê o conteúdo do buffer e apresenta a listagem de /etc/passwd paginadamente.

1 Buffer: é espaço reservado na memória RAM para armazenar algum dado.

Outro exemplo:

```
shell$ cat /etc/passwd | grep bash | wc -l
```

O comando acima determina o número de usuários locais cadastrados para usar o shell /bin/bash.

Outros exemplos:

ls --help | more => apresenta o help paginadamente.

cat /etc/passwd | grep aluno => "cat /etc/passwd | grep -n aluno" mostra o número da linha onde está a ocorrência.

cat /etc/group | sort => apresenta o arquivo ordenadamente.

grep -i ":200:" /etc/passwd | wc -l
=> conta o número de ocorrências do GID (ou UID) 200.

ls -l /etc | grep "^d" | grep "a$"
=> lista os diretórios básicos do /etc cujo nome termina pelo caractere "a".

cut -d: -f1 /etc/passwd | sort => mostra lista ordenada de usernames.

cut -d -f3 /etc/passwd | sort -n => mostra lista ordenada de UID.

cut -d: -f1,3 /etc/passwd | sort => mostra lista ordenada de "usernames:UID".

awk -F: '{print $3, $1}' /etc/passwd | sort -n
=> mostra lista ordenada de "UID usernames".

find /usr -type f -print | wc -l => conta o número total de arquivos regulares no /usr (inclui todos os subdiretórios).

head -12 /etc/passwd | tail -3 => mostra apenas as linhas 10, 11 e 12.

num1=`cat /etc/passwd | wc -l` => conta o número de linhas do arquivo /etc/passwd e guarda na variável $num1.

num2=`echo "$num1 -10 +1" | bc`
=> soma os três números e guarda na variável $num2.

tail -$num2 /etc/passwd | head -3
=> também mostra apenas as linhas 10, 11 e 12.

tail -n +10 /etc/passwd | head -3
=> também mostra apenas as linhas 10, 11 e 12.

5 - REDIRECIONAMENTOS

Os redirecionamentos alteram os fluxos padrão de entrada ou saída de um comando. Normalmente, em linha de comando a **entrada padrão é o teclado**, e a **saída padrão é o monitor**.

5.1 - Saída padrão

Como exemplo de redirecionamento de saída padrão, temos o comando:

```
shell$ ls /etc > /tmp/saida.txt
```

O comando acima faz uma listagem do diretório **/etc** e envia a saída para o arquivo **/tmp/saida.txt**. Se o arquivo não existisse, seria criado. No entanto, se o arquivo existisse, o conteúdo anterior seria sobrescrito.

Caso o arquivo existisse, para manter o conteúdo anterior e acrescentar o novo conteúdo no final do arquivo, o comando seria:

```
shell$ ls /etc >> /tmp/saida.txt
```

NOTA 1: quando se redireciona a saída padrão, esta vai para um arquivo e, portanto, não tem saída no shell [não vem saída para o monitor].

NOTA 2: um efeito semelhante ao redirecionamento de saída padrão pode ser conseguido como o comando **tee**. Por exemplo, "**ls /etc | tee arq.txt**" lista o diretório, envia a saída padrão para o monitor e **também** para o arquivo arq.txt. Caso não quisesse a saída padrão no monitor, bastaria redirecionar para /dev/null: "**ls /etc | tee arq.txt > /dev/null**".

5.2 - Saída de erro

Caso o comando encontrasse algum erro, retornaria uma mensagem de erro. Por exemplo, tentar criar um diretório que já existe gera mensagem de erro. Essa mensagem de erro é chamada de **saída de erro**, e para ser redirecionada precisa incluir o número 2 antes do sinal de maior ">":

```
shell$ mkdir  /etc  2>  /tmp/erro.txt
```

NOTA: se não usar esta notação a saída de erro não é redirecionada e vem para o monitor.

5.3 - Entrada padrão

A entrada padrão é redirecionada com o sinal "<" após o comando. Com isso, o comando irá ler a entrada de um arquivo e não mais do teclado.

Por exemplo, considere o executável que, ao ser disparado pergunta pelo nome e idade, depois finaliza informando o nome e a idade:

```
shell$ nome_idade.sh
        entre com o seu nome: juca
        entre com a sua idade: 286
Seu nome é juca, sua idade é 286 anos
```

Se esse comando fosse disparado com redirecionamento da entrada padrão, não esperaria pelas entradas de teclado:

```
shell$ nome_idade.sh  <  entrada.txt
        entre com o seu nome:
        entre com a sua idade:
Seu nome é juca, sua idade é 286 anos
```

No arquivo **entrada.txt** existem apenas duas linhas, que são as entradas pedidas pelo comando:

```
shell$ more  entrada.txt
juca
286
```

Existem apenas 3 casos para os redirecionamentos:

- saída padrão > ou **1**> ou >> ou **1**>> [pode ser omitido o número 1]
- saída de erro **2**> ou **2**>>
- entrada padrão <

A forma clássica usada no shell para redirecionar de uma única vez tanto a saída de erro quanto a saída padrão para um único arquivo, é:

comando > arquivo_saida 2>&1

NOTA: os redirecionamentos são universais (em qualquer sistema operacional, são idênticos).

6 - EDITOR DE TEXTO vi/vim

No mundo Unix/Linux, o editor de texto padronizado pelo POSIX é o **vi**, cujo nome vem de visual interface. Embora não seja amigável ao iniciante, vale o esforço para aprender, pois em qualquer sistema membro da família Unix ele está presente.

O editor **vim** é o vi "improved" [aprimorado], e deve ser usado sempre que estiver disponível, pois é mais amigável que o vi. Quanto aos comandos, são idênticos.

A primeira noção que se precisa ter do vi/vim é que se trata de um editor de 2 modos, que são:

edição [EDIT ou INSERT]

comando [COMMAND]

Afinal, vi/vim usa apenas o teclado, então num caso teclar estará editando o texto de algum arquivo, no outro estará dando comandos [por exemplo, para salvar o texto].

Para entrar no modo inserção, basta pressionar a letra **i** [de insert]. A partir daí, se está

escrevendo no arquivo.

Para sair do modo inserção e entrar no modo comando, deve comandar a sequência <**ESC**> <**SHIFT**> <**:**>, que são as 3 teclas **ESC+SHIFT+:** e posteriormente pressionar a tecla [ou teclas] com o comando. Nessa situação, a tecla **w** [write] salva o texto no arquivo, a tecla **q** [quit] abandona o editor de texto e volta para o shell.

Exemplo: para criar o arquivo teste.txt e escrever no seu interior, basta seguir a sequência:

1) no shell, comandar "**vi teste.txt**", que abre o editor e toma o shell:

```
shell$ id
uid=1000(aluno) gid=1000(users) grupos=1000(users)
shell$ vi teste.txt
```

```
~
~
~
"teste.txt" [Novo arquivo]
```

2) Pressionar a tecla "**i**" para entrar no modo de inserção [INSERT] de texto;

```
~
~
~
-- INSERÇÃO --
```

3) Escrever o texto "teste de escrita vi";

```
teste de escrita vi
~
~
-- INSERÇÃO --
```

4) Sair do modo de inserção e entrar no modo comando com a sequência de teclado <**ESC**> <**SHIFT**> <**:**>

```
teste de escrita vi
~
~
:
```

5) Pressionar as teclas **wq**, que vai salvar e sair do editor.

Os principais caracteres usados no modo comando são:

x deleta o caractere onde está o cursor;
3x deleta 3 caracteres, o que está sob o cursor e dois à direita;
dw com o cursor posicionado no primeiro caractere da palavra, apaga a palavra inteira;
d4w apaga 4 palavras á direita do cursor;
dd apaga a linha onde está o cursor;
6dd apaga 6 linhas a partir do cursor, para baixo;
u desfaz a edição (undelete);
r substitui o caractere na posição do cursor pelo que for teclado após o **r** (replace);
o entra no modo INSERT ganhando uma linha nova abaixo do cursor;
a entra no modo INSERT ganhando uma caractere à direita do cursor;
q! força a sair do vi sem salvar a edição;
yy copia a linha onde está o cursor. Depois é só colar com **p**;
3yy copia 3 linhas a partir do cursor, para baixo;
ZZ salva e sai do editor (precisa ser em caixa alta).

Para sair do modo INSERT, teclar <**ESC**>.

6.1 - Criação e execução de scripts

Além da linha de comando, os usuários e administradores de sistema costumam usar scripts [roteiros] para automatizar tarefas rotineiras. Desse modo, ao invés de disparar a mesma sequência de comandos todo dia, basta escrever essa sequência num arquivo de texto, definir na primeira linha o nome do interpretador, colocar permissão para execução e disparar o script.

Exemplo 1. Com o vi, criar o arquivo de texto **teste.sh**, que é um shell script:

```
shell$ vi  teste.sh
```

```
#!/bin/bash
echo "Script teste"
~
~
-- INSERÇÃO --
```

Dar permissão padrão de execução para o script:

```
shell$ chmod  755  teste.sh
```

Disparar o executável script:

```
shell$ ./teste.sh
Script teste
```

NOTA: o caminho para o executável pode ser passado de 3 maneiras:

a) Caminho relativo: **./teste.sh**

b) Caminho absoluto: **/home/aluno/teste.sh**

c) Sem passar caminho algum: **teste.sh**

Porém, para executar sem passar caminho algum, precisa incluir na variável de ambiente PATH o diretório aonde está o executável:

```
shell$ echo  $PATH
/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/usr/java/jre1.8.0_77/bin
```

NOTA: neste caso, o diretório /home/aluno não está definido na variável PATH. Então precisa ser adicionado à variável com o comando **export**:

```
shell$ export  PATH=$PATH:/home/aluno
shell$ echo  $PATH
/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:/usr/java/jre1.8.0_77/bin:/home/aluno
```

NOTA: o comando **export** só pode ser usado para o shell baseado em sh. Se fosse csh (C Shell), o comando deveria ser setenv.

Exemplo 2. Com o vi, criar o arquivo de texto **nome_idade.sh**, que é um shell script:

```
shell$ vi  nome_idade.sh
```

```
#!/bin/bash
echo  "Escreva o seu nome: "
read  nome
echo  "Escreva a sua idade: "
read  idade
echo  "Seu nome é $nome, sua idade é $idade"
~
~
-- INSERÇÃO --
```

Dar permissão padrão de execução para o script:

```
shell$ chmod  755  nome_idade.sh
```

Executar o script:

```
shell$ ./nome_idade.sh
Escreva o seu nome: juca
Escreva a sua idade: 286
Seu nome é juca, sua idade é 286
```

Exemplo 3. Com o vi, criar o arquivo de texto **signal.sh**, que é um shell script:

```
shell$ vi  signal.sh;   chmod  755   signal.sh
```

```
#!/bin/bash
comando=$1                                  # $1 é o primeiro parâmetro, que foi passado ao script
if [ -z $comando ]; then                    # -z verifica se a variável $comando está vazia
      echo "ERRO: precisa passar um nome de comando como parâmetro."
      echo "Por exemplo: \"./signal pwd\""
      exit 0
else
      $comando 2> /dev/null
      if [ $? -eq 0 ]; then                 # $? captura o sinal de saída do comando
            echo "Comando $comando executado com sucesso!"
      else
            echo "Houve erro na execução do comando $comando"
      fi
fi
```

Executar este script:

```
shell$ ./signal.sh
ERRO: precisa passar um nome de comando como parâmetro.
Por exemplo: "./signal  pwd"
```

```
shell$ ./signal.sh  pwd
/home/aluno
Comando pwd executado com sucesso!
shell$ ./signal.sh  pdw
Houve erro na execução do comando pdw
```

7 - CONCEITOS DE PROCESSOS

Numa definição simples, processo é o executável enquanto estiver rodando. Do ponto de vista do sistema operacional, processo é uma estrutura responsável pela manutenção das informações necessárias a esta execução.

Os processos são independentes uns dos outros. Cada processo tem seu próprio PID (process identifier), que é um número. Para cada processo, no diretório "/proc" existe um subdiretório com

este número. Os comandos ***ps***, ***pstree*** e ***top*** costumam ser usados para obter informações sobre processos em execução. O comando ***kill*** envia sinais ao processo.

Para controlar a sua ação, um processo possui vários atributos. Esses atributos são:

- **PID**: Process IDentifier - é o número que identifica o processo;
- **PPID**: Parent Process IDentifier - é o processo pai, que gerou um processo filho;
- **UID**: User IDentifier - é o usuário que criou o processo;
- **GID**: Group IDentifier - é o grupo (primário) do usuário que criou o processo;
- **EUID**: Effective User ID - é o usuário efetivo no processo, independente de quem tenha disparado o executável. Ocorre quando se usa o bit ***s*** no campo dono do arquivo: num exemplo, se aluno é o dono desse arquivo e joca tem permissão para executá-lo, a identificação dono do processo (UID) será joca, mas o EUID será aluno;
- **EGID**: Effective Group ID - é o grupo (primário) efetivo no processo, independente de quem tenha disparado o executável. Ocorre quando se usa o bit ***s*** no campo grupo do arquivo: num exemplo, se aluno é o grupo desse arquivo e joca tem permissão para executá-lo, a identificação grupo do processo (GID) será joca, mas o EGID será aluno.

NOTA: para melhor compreensão, ver o anexo EUID/EGID no final deste material.

8 - COMANDOS EM GERAL

A seguir são apresentados comandos de uso geral.

Para obter status de processos, usar o comando **ps**:

```
shell$ ps  -ef
```

Onde a opção **-ef** é para mostrar todos os processos. No Linux, também poderia ser usada a opção **ax** ao invés de -ef.

```
shell$ ps  -ef  |  more
```

Para saber o número total de processos, comandar:

```
shell$ ps  -ef  |  wc  -l
```

Para localizar um (ou mais) processo(s) de nome bash, usar o **grep**:

```
shell$ ps  -ef  |  grep  bash
```

Para localizar um processo pelo número PID, usar a opção **-p**:

```
shell$ ps  -p  PID
```

Onde PID é o número do processo.

O comando que mostra todos os processos numa estrutura de árvore é o **pstree**:

```
shell$ pstree
```

Para saber a ocupação dos sistemas de arquivos montados, usar o comando **df** (disk space usage):

```
shell$ df
```

```
shell$ df  -h
```

Onde a opção -h é para mostrar a ocupação no formato humano (arredonda os tamanhos para K, M, G, etc.)

Para estimar a ocupação de arquivos, usar o comando **du** (file space usage):

```
shell$ du  /home/aluno
```

```
shell$ du -hs /home/aluno
```

Onde as opções -hs são para mostrar a saída no formato humano "**-h**" e apresentar apenas um sumário "**-s**" (total) da ocupação.

Para estimar a ocupação de vários subdiretórios de um mesmo diretório, por exemplo /usr, usar o seguinte **loop for**:

```
shell$ for a in `ls /usr`; do du -hs /usr/$a; done
```

Para facilitar, esta listagem pode ser tanto ordenada numericamente quanto apresentada na forma reversa:

```
shell$ for a in `ls /usr`; do du -s /usr/$a; done | sort -nr
2598284         /usr/share
2524636         /usr/lib
395964          /usr/bin
52704           /usr/sbin
132             /usr/local
4               /usr/etc
```

Para saber há quanto tempo o servidor está rodando, usar o comando **uptime**:

```
shell$ uptime
```

O comando uptime também mostra a carga média (**load average**) do sistema, tomados em três medidas, que são médias de 1, 5 e 15 minutos. O load average está relacionado ao número médio de processos prontos para rodar, rodando ou aguardando acesso a I/O (input/output), por CPU (ou núcleo). Se o load average for próximo a zero, o sistema está ocioso, se for 6 e houver 4 CPUs, na média significa que 2 processos estão esperando pela CPU.

NOTA 1: load average alto não necessariamente significa alto consumo de CPU. Como exemplo, pode haver uma longa fila de processos aguardando I/O, e devido a isso estes processos não usam a CPU.

NOTA 2: se o load average estiver baixo, mesmo com o consumo de CPU em 100%, isto significa que de um modo geral não está havendo gargalos nem espera de recursos pelos usuários.

Para saber o total de memória RAM e SWAP livre, usar o comando **free**:

```
shell$ free
               total        used        free      shared  buff/cache   available
Mem:         8050676     1377492     5194896      479828     1478288     5899952
Swap:        8190972           0     8190972
```

NOTA 1: swap é um espaço de armazenamento para processos. Normalmente este espaço está em disco. O processo no swap não ganha diretamente a CPU.

NOTA 2: o uso constante de swap indica problemas de dimensionamento do sistema (tem pouca RAM ou processos demais). Muito uso de swap degrada o desempenho do sistema, pois a velocidade de acesso a disco é muito inferior ao da RAM.

Para saber quais processos estão consumindo mais recursos computacionais, usar o comando **top**:

```
shell$ top
```

NOTA: o comando **top** mostra em tempo real uma lista de processos. Esta lista de processos pode ser apresentada ordenada por consumo de CPU, pelo uso de memória, etc. Para sair do top, teclar **q** ou <CTRL>c.

Para obter uma lista dos últimos comando executados no shell, usar o comando **history**:

```
shell$ history
```

NOTA: nem todo shell implementa o histórico de comandos. Os principais (bash, ksh, sh, etc) fazem isso.

Para obter uma lista com todas as variáveis do ambiente do usuário, usar o comando **env**:

```
shell$ env
```

Para carregar um valor numa variável, usar o comando **export**:

```
shell$ export  TESTE=aluno
shell$ echo  $TESTE
aluno
```

NOTA 1: ao carregar um valor numa variável que já exista, o valor anterior será perdido. Para excluir uma variável, usar o comando **unset**. Por exemplo, ***unset TESTE***.

NOTA 2: se estiver usando C shell (ou um derivado deste, como tcsh), o comando é **setenv** e não export.

Para criar um linque simbólico (symbolic ou **soft link**), usar o comando **ln** com a opção **-s**:

```
shell$ ln  -s  passwd2  /etc/passwd
shell$ ls  -l  passwd2
lrwxrwxrwx  1  aluno  aluno  11  Ago  5 12:59   passwd2  ->  /etc/passwd
```

NOTA: o linque simbólico é um mero "atalho" que aponta para o arquivo físico. Pode apontar para um arquivo em outro sistema de arquivo.

Para criar um linque físico (**hard link**), usar o comando **ln**:

```
shell$ echo  "teste"  >  teste.txt
shell$ ln  teste.txt  hard-teste.txt
shell$ ls  -li  teste.txt  hard-teste.txt
21772860  -rw-rw-r--  2  aluno  aluno  6  Ago  5  13:05  hard-teste.txt
21772860  -rw-rw-r--  2  aluno  aluno  6  Ago  5  13:05  teste.txt
```

NOTA 1: um **hard link** é um nome de arquivo que aponta para o mesmo i-node de outro arquivo. Não pode ser criado entre diferentes sistemas de arquivo.

NOTA 2: ao contrário do soft link, pode ser excluído o arquivo físico contra o qual foi criado o hard link, que permanece o acesso ao arquivo. Na prática, excluir um arquivo é excluir o último hard link que aponta para o i-node deste arquivo.

NOTA 3: a opção "**-i**" no comando **ls** é para mostrar o número do i-node do arquivo.

9 - PROCESSO DE BOOT E RUNLEVEL OU TARGET

Quando a máquina é ligada, o sistema BIOS[2] carrega e executa o código do boot loader no MBR, que é o Master Boot Record ou Registro Mestre de Boot. Boot loader é quem inicia o processo de carregamento do kernel do sistema operacional.

É o MBR que aponta para alguma partição com setor de boot, que na sequência carrega o sistema operacional. No Linux, exemplos de boot loaders são LILO e GRUB. LILO significa LInux LOader e GRUB é GRand Unified Bootloader.

Depois do kernel completamente carregado é executado o /sbin/init, que é o primeiro processo iniciado, portanto tem PID[3] igual a 1. O processo init faz algumas verificações básicas, como integridade do sistema de arquivos, e na sequência desencadeia a inicialização de processos necessária tanto para que o sistema operacional funcione corretamente, quanto para iniciar os serviços que o sistema sustenta.

Originalmente, o desencadeamento desta sequência de inicialização de processos era feita pelo SystemV (System five), de 1983. O SystemV ainda é usado atualmente, mas nas distribuições Linux está sendo gradativamente substituído pelo SystemD. SystemD é um novo sistema de inicialização, adotado pela primeira vez no ano de 2011 pelo Fedora.

Depois do Fedora, as principais distribuições Linux também passaram a adotar o SystemD:

Distribuição	Ano
Red Hat 7	2014
CentOS 7	2014
Ubuntu 15	2015
Debian 8	2015

9.1 - SystemV e runlevel

No SystemV, o desencadeamento de processos de inicialização está baseado em runlevels (níveis de execução). Existem 7 runlevels definidos , que vão de 0 a 6.

Ao final do carregamento do kernel, o processo **init** (PID 1) é iniciado. É o **init** que inspeciona o arquivo /etc/inittab para determinar o modo de operação ou runlevel.

Para saber se o sistema operacional é SystemV, procurar pelo processo de PID número 1:

2 BIOS: Basic Input/Output System [Sistema Básico de Entrada/Saída] é um programa de computador pré-gravado em memória permanente [firmware] executado por um computador quando ligado.

3 PID: Process Identifier ou identificador do processo, que é um número inteiro atribuído a cada processo no sistema.

```
shell$ ps -p 1
  PID  TTY      TIME CMD
    1  ?    00:00:00 init
```

NOTA: se fosse SystemD, o processo de PID número 1 seria o **systemd**.

O arquivo /etc/inittab normalmente tem as seguintes configurações:

```
shell$ more /etc/inittab
# Default runlevel. The runlevels used by RHS are:
# 0 - halt (Do NOT set initdefault to this)
# 1 - Single user mode
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
# 3 - Full multiuser mode
# 4 - unused
# 5 - X11
# 6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:                # Console Text Mode
# id:5:initdefault:              # Console GUI Mode
```

Com base no /etc/inittab acima, esse sistema irá iniciar no runlevel 3, que é o modo texto sem interface gráfica.

Baseado na seleção de runlevel, o processo init passa a executar os scripts[4] de inicialização localizados nos subdiretórios do diretório /etc/rc.d/. Scripts de inicialização são scripts que iniciam processos ou serviços.

No diretório /etc/rc.d existem os subdiretórios /etc/rc.d/rc0.d/, /etc/rc.d/rc1.d/ e assim por diante, até /etc/rc.d/rc6.d/. Cada um desses subdiretórios corresponde a um runlevel.

A descrição de cada runlevel segue abaixo:

Modo	Diretório do runlevel	Descrição
0	/etc/rc.d/rc0.d	halt, desliga o sistema
1	/etc/rc.d/rc1.d	single user mode, ou modo de manutenção [sem usuários nem serviços]. Além de 1, também pode ser ***s*** ou ***S***
2	/etc/rc.d/rc2.d	não é usado [mas poderia ser definido algum uso particular pelo usuário]. Porém, este é o default do Debian

4 script: um script é um roteiro, que contém comandos ou mesmo instruções em linguagem de programação específica do interpretador do script em específico. Um script é um arquivo de texto. Exemplo: shell script, que contém comandos shell ou instruções na sintaxe de programação shell.

3	/etc/rc.d/rc3.d	full multi-user mode, ou modo completo de serviço sem interface gráfica
4	/etc/rc.d/rc4.d	reservado, portanto não é usado [mas poderia ser definido algum uso particular pelo usuário]
5	/etc/rc.d/rc5.d	full multi-user mode, ou modo completo de serviço com interface gráfica
6	/etc/rc.d/rc6.d	reboot, reinicia o sistema (*shutdown -r*)

Para dar um boot e ir para um determinado runlevel, basta usar o comando **init** seguido do número do runlevel. Por exemplo, para ir para o modo de manutenção, comandar:

```
root# init  1
```

Depois do boot, vai mostrar:

```
Give root password for maintenance
(or type Control-D to continue):
root# runlevel
1  S
root# exit
```

Em cada um dos diretórios do runlevel existem linque simbólicos que apontam para os respectivos scripts de inicialização. A notação adotada é usar o "**S**" para iniciar o serviço no runlevel específico e um "**K**" para parar o serviço quando desligar ou reiniciar o sistema.

Por exemplo, num Red Hat 6 (ou CentOS 6), o diretório /etc/rc.d/rc3.d mostra:

```
shell$ ls  -l  /etc/rc.d/rc3.d
...
lrwxrwxrwx  1 root root 18 Ago 11  2013      K85ebtables -> ../init.d/ebtables
lrwxrwxrwx  1 root root 14 Abr   5 18:03     K88sssd -> ../init.d/sssd
lrwxrwxrwx  1 root root 15 Ago   7  2015     K89rdisc -> ../init.d/rdisc
lrwxrwxrwx  1 root root 14 Jun 20 21:47     K91capi -> ../init.d/capi
lrwxrwxrwx. 1 root root 14 Set  15  2010     K99rngd -> ../init.d/rngd
lrwxrwxrwx  1 root root 17 Ago   5  2015     S01sysstat -> ../init.d/sysstat
lrwxrwxrwx  1 root root 22 Fev 11  2016      S02lvm2-monitor -> ../init.d/lvm2-monitor
lrwxrwxrwx  1 root root 16 Nov 17  2012      S07iscsid -> ../init.d/iscsid
lrwxrwxrwx  1 root root 19 Ago   7  2015     S08ip6tables -> ../init.d/ip6tables
lrwxrwxrwx  1 root root 18 Ago   7  2015     S08iptables -> ../init.d/iptables
...
```

Para saber o runlevel atual e anterior, comandar **runlevel**:

```
shell$ runlevel
N 5
```

Com base na saída do comando acima, sabemos que o sistema agora está no runlevel 5 (interface gráfica), e que isso não mudou desde o último boot. Por exemplo, se anteriormente estivesse no runlevel 3 e depois fosse reconfigurado para runlevel 5, depois do boot a saída do comando seria "**3 5**" ao invés de "**N 5**".

Numa máquina SystemV, neste exemplo um Debian 7, para determinar os runlevels de finalização (**K**) e de inicialização (S) do serviço rsyslog, comandar:

```
shell$ ls -l /etc/rc?.d | grep -B 20 rsyslog | grep -e '/etc/rc\|rsyslog'
/etc/rc0.d:
lrwxrwxrwx 1 root root 17 Jul 21 20:38 K04rsyslog -> ../init.d/rsyslog
/etc/rc1.d:
lrwxrwxrwx 1 root root 17 Jul 21 20:38 K04rsyslog -> ../init.d/rsyslog
/etc/rc2.d:
lrwxrwxrwx 1 root root 17 Jul 21 20:37 S16rsyslog -> ../init.d/rsyslog
/etc/rc3.d:
lrwxrwxrwx 1 root root 17 Jul 21 20:37 S16rsyslog -> ../init.d/rsyslog
/etc/rc4.d:
lrwxrwxrwx 1 root root 17 Jul 21 20:37 S16rsyslog -> ../init.d/rsyslog
/etc/rc5.d:
lrwxrwxrwx 1 root root 17 Jul 21 20:37 S16rsyslog -> ../init.d/rsyslog
/etc/rc6.d:
lrwxrwxrwx 1 root root 17 Jul 21 20:38 K04rsyslog -> ../init.d/rsyslog
```

NOTA: no comando **ls** acima, o caractere "**?**" é curinga, e pode substituir qualquer outro. No primeiro **grep**, a opção "**-B**" é para listar 20 linhas acima (before) do padrão encontrado. No segundo **grep**, a opção "**-e**" é para procurar por expressão regular, e no caso acima trata-se de buscar um arquivo ou o outro.

No Debian, para remover os linques ou adicionar os linques de inicialização, usar o comando **update-rc.d**. Neste exemplo, o serviço é o rsyslog:

```
root# update-rc.d rsyslog remove
```

Para adicionar os linques simbólicos na inicialização, comandar:

```
root# update-rc.d  rsyslog  defaults
```

NOTA: os *defaults* estão no próprio script de inicialização do serviço, no diretório /etc/init.d:

```
shell$ head  /etc/init.d/rsyslog
#! /bin/sh
### BEGIN INIT INFO
# Provides:          rsyslog
# Required-Start:    $remote_fs  $time
# Required-Stop:     umountnfs  $time
# X-Stop-After:      sendsigs
# Default-Start:     2  3  4  5
# Default-Stop:      0  1  6
# Short-Description: enhanced syslogd
# Description:       Rsyslog is an enhanced multi-threaded syslogd.
```

Num Red Hat 6 (ou CentOS), o comando para criar ou remover os linques de inicialização é **chkconfig**. Por exemplo:

```
root# chkconfig  --list  rsyslog
root# chkconfig  --del  rsyslog
root# chkconfig  --add  rsyslog
```

Do mesmo modo que no Debian, os scripts de inicialização também ficam no diretório /etc/init.d.

Os scripts no diretório /etc/init.d são usados para gerenciar os serviços. Por exemplo, para saber o status do serviço rsyslog, comandar:

```
root# /etc/init.d/rsyslog  status
 [ ok ] rsyslogd is running.
```

Para parar o serviço, usar **stop**, para iniciar usar **start**:

```
root# /etc/init.d/rsyslog  stop
 [ ok ]  Stopping enhanced syslogd: rsyslogd.
root# /etc/init.d/rsyslog  start
 [ ok ]  Starting enhanced syslogd: rsyslogd.
```

9.2 - SystemD e targets

Para saber se o sistema operacional é SystemD, procurar pelo processo de PID número 1:

```
shell$ ps -p 1
  PID  TTY      TIME   CMD
    1  ?    00:00:01   systemd
```

No SystemD, o conceito de runlevels do SystemV foi substituído por targets (alvos), que têm semelhança com os runlevels. A tabela abaixo mostra esta correspondência:

Runlevel	Unidades de Targets	Descrição
0	runlevel0.target, poweroff.target	halt, desliga o sistema
1	runlevel1.target, rescue.target	single user mode, ou modo de manutenção
2	runlevel2.target, multi-user.target	multi usuário, modo texto
3	runlevel3.target, multi-user.target	multi usuário, modo texto
4	runlevel4.target, multi-user.target	multi usuário, modo texto
5	runlevel5.target, graphical.target	Mult usuário, com interface gráfica
6	runlevel6.target, reboot.target	reboot, reinicia o sistema

Porém, o comando a ser usado agora é o **systemctl**.

Por exemplo, o equivalente a runlevel é:

```
root# systemctl list-units --type target
```

O comando acima mostra as unidades de target correntemente carregadas.

Para saber qual unidade de target está sendo usada no momento, usar o comando:

```
root# systemctl get-default
graphical.target
```

Para alterar a target default de graphical.target para mult-user, comandar:

```
root# systemctl  set-default  multi-user.target
Removed symlink /etc/systemd/system/default.target.
Created symlink from /etc/systemd/system/default.target to /usr/lib/systemd/system/multi-user.target.
root# systemctl  get-default
multi-user.target
```

Para retornar o default para graphical.target, comandar:

```
root# systemctl  set-default  graphical.target
Removed symlink /etc/systemd/system/default.target.
Created symlink from /etc/systemd/system/default.target to /usr/lib/systemd/system/graphical.target.
root# systemctl  get-default
graphical.target
```

Reparar que agora os linques simbólicos ficam no diretório **/etc/systemd/system**, e as unidades (units) ficam em **/usr/lib/systemd**. Em SystemD, uma unidade (unit) se refere a qualquer recurso que o sistema saiba como gerenciar, recurso este que é definidos pelo uso de arquivos de configurações chamados de arquivos de unidades.

O arquivo de unidade SystemD é equivalente ao script de inicialização do SystemV, que fica em /etc/init.d/.

Por exemplo, num Red Hat 7, o arquivo de unidade do serviço rsyslog é:

```
shell$ head  /usr/lib/systemd/system/rsyslog.service
[Unit]
Description=System Logging Service
;Requires=syslog.socket
After=network.target

[Service]
Type=notify
EnvironmentFile=-/etc/sysconfig/rsyslog
ExecStart=/usr/sbin/rsyslogd -n $SYSLOGD_OPTIONS
Restart=on-failure
```

Ainda seguindo o exemplo, para saber o status do serviço rsyslog, comandar:

```
root# systemctl  status  rsyslog
  rsyslog.service - System Logging Service
  Loaded: loaded (/usr/lib/systemd/system/rsyslog.service; enabled; vendor preset: enabled)
  Active: active (running) since Qui 2017-09-07 16:06:39 -03; 5h 2min ago
 Main PID: 1363 (rsyslogd)
  CGroup: /system.slice/rsyslog.service
          └─1363 /usr/sbin/rsyslogd -n

Set 07 16:06:39 pinguim systemd[1]: Starting System Logging Service...
Set 07 16:06:39 pinguim systemd[1]: Started System Logging Service.
```

Para parar o serviço usar **stop**, para iniciar usar **start**:

```
root# systemctl  stop  rsyslog
root# systemctl  start  rsyslog
```

Outros exemplos de comandos são:

systemctl poweroff	=> init 0
systemctl rescue (systemctl rescue.target)	=> init 1 (s, S)
systemctl isolate runlevel2.target	=> init 2
systemctl isolate runlevel3.target	=> init 3
systemctl isolate runlevel4.target	=> init 4
systemctl isolate runlevel5.target	=> init 5
systemctl reboot (systemctl reboot.target)	=> init 6

10 - PROCESSO DE LOGON

Para logar no sistema, o usuário precisa estar previamente cadastrado. Se for um usuário local, terá uma entrada no arquivo **/etc/passwd**, que é o catálogo ou cadastro de usuários locais. Cada usuário local é uma linha no arquivo, por exemplo o usuário root tem a seguinte linha:

```
shell$ grep  root  /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

Cada linha do /etc/passwd tem sete campos:

1° campo é o ***username***: root;
2° campo tem um ***x***, pois a *hash*[5] de senha [digest] está no arquivo /etc/shadow [Linux];
3° campo é o ***UID*** [**U**ser **ID**entifier]: 0;
4° campo é o ***GID*** [**G**roup **ID**entifier]: 0;
5° campo é a ***descrição*** do usuário: root;
6° campo é a ***home*** do usuário: /root;
7° campo é um comando (executável), que neste caso é o ***shell*** do usuário: /bin/bash.

Para o usuário aluno, tem a seguinte linha:

```
shell$ grep  aluno  /etc/passwd
aluno:x:1000:1000:Aluno:/home/aluno:/bin/bash
```

Onde: o **username** é aluno, **UID** é 1000, **GID** é 1000, **descrição** é "Aluno", a **home** é /home/aluno e o **shell** é /bin/bash.

Os **GID** estão cadastrados no arquivo **/etc/group**, que é o catálogo ou cadastro de grupos locais.

Para o grupo de GID 1000 tem a seguinte entrada:

```
shell$ grep  ":1000:"  /etc/group
aluno:x:1000:
```

Onde os campos são:

1° campo é o ***groupname***: aluno;
2° campo *x*, indica que a senha de grupo (se houver) está em /etc/gshadow [Linux];
3° campo é o **GID**: 1000;
4° campo é para incluir os usernames que participam como grupo secundário.

Um usuário participa de apenas um grupo primário, ou seja, tem um único GID. Mas pode participar de vários grupos secundários. Por exemplo, uma busca pelo username aluno no /etc/group mostra:

```
shell$ grep  aluno  /etc/group
sys:x:3:aluno
mail:x:12:postfix,dovecot,aluno
aluno:x:1000:
```

5 Hash: é o mesmo que *message digest*, que é uma função que produz um código de tamanho fixo como saída para uma dada mensagem de entrada [a senha].

Na primeira e na segunda linha, o **username** aluno aparece no 4° campo, portanto aluno participa dos grupos secundários sys e mail. Na terceira linha, é o ***groupname*** aluno que aparece no 1° campo.

Para aluno, o comando **id** vai mostrar a seguinte saída:

```
shell$ id
uid=1000(aluno) gid=1000(aluno) grupos=1000(aluno),3(sys),12(mail)
```

O usuário também precisa ter uma entrada no arquivo de senhas, que no Linux é o **/etc/shadow**. Por exemplo:

```
root# grep  aluno  /etc/shadow
aluno:$1$GXJzit5J$vS4wC8AW6hV8zvLu6Dtxc.:17213::::::
```

Onde os campos são:

1° campo é o ***username***: aluno;
2° campo é ***hash de senha*** [no caso, MD5 digest];
3° campo é o dia da alteração desta senha [desde o 1° de janeiro de 1970]: 17213;
4° campo em diante: se não vazios, servem para aplicar a política de senha e sua expiração.

O usuário então cadastrado e com uma senha válida pode logar no sistema. Ao logar, receberá o seu **shell**, que é a interface usual e que permite interagir com o sistema em linha de comando.

Nesse processo, o usuário também carrega as suas variáveis de ambiente. Por exemplo, $SHELL, $HOME, $PATH.

11 - PERMISSÕES DO SISTEMA DE ARQUIVOS

As permissões do sistema de arquivos definem o acesso aos arquivos. Por acesso, nesse caso entendemos leitura, escrita e execução, entre outros. Para alterar estes acessos, existem três comandos: ***chmod***, ***chown*** e ***chgrp***.

Para entender o conceito de permissão de acesso ao sistema de arquivos, antes é necessário ter em mente que:

- Todo arquivo tem um dono. O dono é quem cria o arquivo;
- Todo arquivo pertence a um grupo primário de usuários. O conceito de grupo envolve usuários com atividades que requerem um conjunto semelhante de permissões de acesso a arquivos;
- Aqueles usuários que não são o próprio dono do arquivo ou não pertencem ao grupo do dono, são considerados "outros" do ponto de vista do acesso ao arquivo;
- Permissões são concessões no acesso a arquivos. A nível de sistema de arquivo, usa-se [por exemplo] os bits ***r*** [read – leitura], ***w*** [write – escrita] e ***x*** [execução ou acesso a diretório].

Para as permissões, existem 3 campos distintos: o do dono, do grupo e outros.

Por exemplo, o comando ***ls -l /home/aluno*** mostra:

```
shell$ ls -l /home/aluno
-rw-rw-r--    1    aluno   aluno   59939    Ago 18 21:13    arq.txt
drwxr-xr-x    2    aluno   aluno    4096    Mai 12 09:15    dir1
```

Olhando o resultado dessa listagem no formato longo, identificamos que o último campo de cada linha é o nome do arquivo, que já aparecia na listagem **ls** [sem formato longo]. Mas que agora apresenta mais atributos para cada arquivo.

Uma visão detalhada desses atributos mostra que:

-rw-rw-r-- drwxr-xr-x permissões do sistema de arquivo	1 2 nº de linques para o arquivo	aluno aluno username (dono do arquivo)	aluno aluno groupname (grupo do arquivo)	59939 4096 tamanho do arqui-vo (em bytes)	Ago 18 21:13 Mai 12 09:15 data da última edição no arquivo	arq.txt dir1 nome do arquivo

No entanto, o que foi chamado de permissões precisa ter descontado o primeiro caractere, que designa o tipo do arquivo.

O caractere "**d**" [por exemplo, **d**rwxr-xr-x] designa que o tipo do arquivo é um **diretório**.

Se iniciar com um traço "-" [por exemplo, -rw-rw-r--], o arquivo é do tipo **regular** ou ordinário.

Se iniciar com um caractere **"l"** [por exemplo, **l**rwxrwxrwx], trata-se de **soft ink** [linque

simbólico].

Dos nove bits de permissões (por exemplo, rw-rw-r--), os três primeiros são permissões do dono, os três seguintes são do grupo e os três últimos dos outros. Não existe sobreposição dessas permissões. Por exemplo, as permissões dadas ao grupo não afetam as permissões de acesso para o dono do arquivo.

-**rw**-rw-r--	-rw-**rw**-r--	-rw-rw-**r**--
dono do arquivo	grupo do arquivo	outros

11.1 - chmod

O comando ***chmod*** altera as permissões de acesso ao arquivo. Pode ser alterado no modo numérico [absoluto ou octal] ou simbólico.

Para poder alterar as permissões, o usuário precisa ser dono do arquivo, ou então ser root.

11.1.1 - Modo numérico ou absoluto [octal]

O modo numérico usa a notação octal para traduzir os bits de permissão dos arquivos em números de 0 a 7. A tabela abaixo mostra a relação entre bits de permissão e números.

rwx	binário	octal
---	000	0
--x	001	1
-w-	010	2
-wx	011	3
r--	100	4
r-x	101	5
rw-	110	6
rwx	111	7

Convém notar que o bit ***r*** está associado ao octal **4**, ***w*** ao **2** e ***x*** ao **1**:

r => 4
w => 2
x => 1

Notar também que os valores octais são cumulativos. Por exemplo, a permissão "**r-x**" é 4+0+1=**5**, "**rwx**" é 4+2+1=**7**.

Exemplos:

```
shell$ chmod 444 arq.txt
shell$ ls -l arq.txt
-r--r--r--    1 aluno aluno    59939  Ago 18 21:13 arq.txt
shell$ chmod 246 arq.txt
shell$ ls -l arq.txt
--w-r--rw-    1 aluno aluno    59939  Ago 18 21:13 arq.txt
shell$ chmod 750 dir1
shell$ ls -ld dir1
drwxr-x---    1 aluno aluno     4096  Mai 12 09:15 dir1
```

NOTA: o bit ***x*** em arquivo regular é permissão de execução, já em diretório dá acesso ao seu conteúdo. Afinal, o diretório é estrutura organizacional e não pode ser executado.

11.1.2 - Modo simbólico

No modo simbólico, a alteração de permissão de arquivos segue o esquema:

chmod [ugoa][+−=][rwxst]

A notação usada no modo simbólico é a seguinte:

- Os caracteres [**ugoa**] designam as permissões do dono [**u**, user], do grupo [**g**, group], dos outros [**o**, others] e de todos [**a**, all];
- Os símbolos [+−=] definem a seleção para as permissões, onde o sinal [+] acrescenta a permissão às que já existem no arquivo, o sinal [−] subtrai das existentes e o sinal [=] remove todas as permissões que haviam no arquivo e impõem a que está sendo alterada para o(s) usuário(s);
- Os bits [**rwxst**] são os bits de permissão a serem alterados no arquivo.

Exemplos:

```
shell$ ls -l arq.txt
--w-r--rw-    1 aluno aluno    59939  Ago 18 21:13  arq.txt
shell$ chmod u+rx arq.txt
shell$ ls -l arq.txt
-rwxr--rw-    1 aluno aluno    59939  Ago 18 21:13  arq.txt
shell$ chmod g-r arq.txt
shell$ ls -l arq.txt
-rwx---rw-    1 aluno aluno    59939  Ago 18 21:13  arq.txt
shell$ chmod a+x arq.txt
shell$ ls -l arq.txt
-rwx--xrwx    1 aluno aluno    59939  Ago 18 21:13  arq.txt
shell$ chmod ug=r arq.txt
shell$ ls -l arq.txt
-r--r--rwx    1 aluno aluno    59939  Ago 18 21:13  arq.txt
```

As permissões poderiam ser complexas e alterar simultaneamente mais de uma categoria de usuários, mas isso não é simples de ser feito no modo simbólico:

```
shell$ ls -l arq.txt
-r--r--rwx    1 aluno aluno    59939  Ago 18 21:13  arq.txt
shell$ chmod u+w,o-wx arq.txt
shell$ ls -l arq.txt
-rw-r--r--    1 aluno aluno    59939  Ago 18 21:13  arq.txt
shell$ chmod a+w,o-rw arq.txt
shell$ ls -l arq.txt
-rw-rw----    1 aluno aluno    59939  Ago 18 21:13  arq.txt
```

Para os diretórios, o procedimento é o mesmo:

```
shell$ ls -ld dir1
drwxr-x---    1 aluno aluno     4096  Mai 12 09:15  dir1
shell$ chmod a=rx dir1
shell$ ls -ld dir1
dr-xr-xr-x    1 aluno aluno     4096  Mai 12 09:15  dir1
```

11.1.3 - Bits *s* e *t*

Além dos bits de permissão ***r***, ***w*** e ***x***, também são frequentemente usados os bits ***s*** e ***t***.

11.1.3.1 - Bit *s*

O bit ***s*** em arquivos regulares pode estar tanto no campo do dono do arquivo quanto do grupo. Por exemplo: **-rwsr-x--x**, **-rwxr-s--x**, **-rwsr-s--x**. Em arquivos regulares, o bit ***s*** é também um bit de execução.

Se o bit ***s*** aparecer no campo do **dono** do arquivo é **SUID** [Set User ID], se aparecer no campo do **grupo** é **SGID** [Set Group ID]. Esse bit altera a identificação do usuário (ou do grupo) no processo gerado pela execução do arquivo executável.

Se for SUID, atribui a permissão do processo como idêntica a do dono do arquivo, independentemente de quem tenha executado.

Se for SGID, atribui a permissão do processo como idêntica a do grupo do arquivo, independentemente de quem tenha executado.

Por exemplo, o executável usado para alterar senha é o comando **passwd**, que tem as seguintes permissões:

```
shell$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 22520 Fev 26 2009 /usr/bin/passwd
shell$ ls -l /etc/shadow
-r-------- 1 root root 1324 Out 12 21:00 /etc/shadow
```

No caso, o dono do executável **passwd** é o root, porém este permite que qualquer usuário execute o comando, como pode ser visto pelo bit **x** no campo outros.

Porém, quando alguém executa o comando **passwd**, também precisa escrever a nova **hash** de senha no arquivo **/etc/shadow**, que não tem permissão de escrita para outros [por questões de segurança, obviamente].

Então, o bit ***s*** no campo do dono do executável **passwd** faz com que o processo originado por qualquer usuário tenha privilégio de acesso do dono do arquivo, isto é, de root, que consegue escrever no **/etc/shadow**. Desse modo, o usuário consegue alterar a sua própria senha.

Para impor o bit SUID num arquivo, usar o **comando** chmod com a notação octal abaixo:

```
shell$ id
uid=1000(aluno) gid=1000(aluno) grupos=1000(aluno),3(sys),12(mail)
shell$ pwd
/tmp
shell$ cp  /bin/mkdir  mkdir2
shell$ ls  -l  mkdir2
-rwxr-xr-x  1  aluno  aluno  43504  Abr  11 14:37  mkdir2
shell$ chmod  4755  mkdir2
shell$ ls  -l  mkdir2
-rwsr-xr-x  1  aluno  aluno  43504  Abr  11 14:37  mkdir2
```

NOTA: o comando chmod, no modo octal, recebe um parâmetro com 4 dígitos. Porém, se o[s] primeiro[s] bit[s] for[em] 0 [zero], este[s] pode[m] ser omitido[s].

Exemplos:

chmod 0123 arq.txt é idêntico a ***chmod 123 arq.txt***
chmod 0023 arq.txt é idêntico a ***chmod 23 arq.txt***

Para verificar o funcionamento desse bit será necessário outro usuário, que não pertença ao grupo de aluno. Neste caso, se ainda não existe, adicionar o usuário juca. Para adicionar um novo usuário, precisa ser **root**:

```
root# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
root# /usr/sbin/useradd -m -s /bin/bash -c "usuario juca" -d /home/juca juca
root# tail -1 /etc/passwd
juca:x:1001:1001:usuario juca:/home/juca:/bin/bash
root# passwd juca
Mudando senha para o usuário juca.
Nova senha:
```

Agora, o usuário juca loga em outro shell [por exemplo, <CTRL><ALT>F3], e usa o executável em **/tmp/mkdir2** para criar um novo diretório:

```
juca$ id
uid=1001(juca) gid=1001(juca) grupos=1001(juca)
juca$ cd /tmp
juca$ /tmp/mkdir2 juca
juca$ ls -ld /tmp/juca
drwxr-xr-x 2 aluno juca 4096 Abr 11 15:00 /tmp/juca
```

NOTA: repare que o diretório criado por juca pertence a **aluno** e não a juca.

Analogamente ao caso acima, se o dono do arquivo [aluno] mudar as permissões para ***-rwxr-sr-x*** (SGID) com o comando **chmod 2755 /tmp/mkdir2**, quando juca criar outro diretório este será do grupo de aluno [aluno].

Do mesmo modo, se aluno usar o comando **chmod 6755 /tmp/mkdir2**, irá alterar as permissões do arquivo **/tmp/mkdir2** para ***-rwsr-sr-x***. Nesse caso, se juca usar o comando /tmp/mkdir para criar outro diretório, este pertencerá a aluno e ao grupo aluno [que é o grupo de aluno].

11.1.3.2 - Bit *t*

O bit ***t*** em diretórios dá a permissão append-only [sticky bit], e é usado em diretórios públicos como o /tmp.

Se não houvesse o bit ***t*** em diretórios com permissão de escrita para outros usuários, quem criasse arquivos neste diretório efetivamente não seria o dono, e poderia ter seus arquivos removidos. Por exemplo, se aluno criasse um arquivo neste diretório, juca poderia excluir o arquivo de aluno. Com o uso do bit ***t***, quem cria o arquivo efetivamente é o dono.

No exemplo abaixo, o usuário root irá criar um novo diretório público de nome **/tmp2**, e nele adicionar o bit ***t***, com permissões 1777:

```
root# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
root# ls -ld /tmp
drwxrwxrwt 28 root root 4096 Abr 10 12:00 /tmp
root# mkdir /tmp2
root# ls -ld /tmp2
drwxr-xr-x 2 root root 4096 Abr 11 15:00 /tmp2
root# chmod 1777 /tmp2
drwxrwxrwt 2 root root 4096 Abr 11 15:00 /tmp2
```

11.2 - chown

O comando **chown** é usado pelo root para alterar a posse do arquivo. Por exemplo:

```
root# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
root# ls -l /tmp/teste.txt
-rw-r--r--  1  root  root  43504  Abr  11 14:37  /tmp/teste.txt
root# chown aluno /tmp/teste.txt
root# ls -l /tmp/teste.txt
-rw-r--r--  1  aluno  root  43504  Abr  11 14:37  /tmp/teste.txt
```

Além de alterar a posse, também poderia alterar o grupo primário. Por exemplo:

```
root# touch /tmp/arq.txt
root# ls -l /tmp/arq.txt
-rw-r--r--  1  root  root  0  Ago  15  15:32  /tmp/arq.txt
root# chown aluno:aluno /tmp/arq.txt
root# ls -l /tmp/arq.txt
-rw-r--r--  1  aluno  aluno  0  Ago  15  15:32  /tmp/arq.txt
```

Uma situação real da necessidade de uso do comando **chown**, é quando o root copia um arquivo para algum usuário. Neste caso, precisa alterar a posse, senão permanece como pertencente ao root:

```
root# cp /etc/shadow /home/aluno/shadow2
root# ls -l /home/aluno/shadow2
-r--------  1  root  root  1732  Set  8 16:09  /home/aluno/shadow2
root# chown aluno:aluno /home/aluno/shadow2
root# ls -l /home/aluno/shadow2
-r--------  1  aluno  aluno  1732  Set  8 16:09  /home/aluno/shadow2
```

11.3 - chgrp

O comando **chgrp** é usado pelo root para alterar o grupo do arquivo. Por exemplo:

```
root# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
root# ls -l /tmp/teste.txt
-rw-r--r--  1  root  root  43504  Abr  11 14:37  /tmp/teste.txt
root# chgrp aluno /tmp/teste.txt
root# ls -l /tmp/teste.txt
-rw-r--r--  1  root  aluno  43504  Abr  11 14:37  /tmp/teste.txt
```

12 - GERENCIAMENTO DE USUÁRIOS

12.1 - Criação de usuário e grupos

Para criar o usuário juca, antes precisa ganhar privilégios de root:

```
shell$ sudo su -
[sudo] password for aluno:
root# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

Para saber se já existe o usuário juca, podem ser usados dois comandos, **grep** ou **id**:

```
root# grep juca /etc/passwd
```

```
root# id juca
id: juca: no such user
```

O comando para criar um usuário local é **useradd**:

```
root# /usr/sbin/useradd -m -s /bin/bash juca
```

No comando acima, a opção **-m** é para criar a home do usuário e **-s** especifica o shell. Como não foi feito referência a grupo, por *default* foi criado o grupo juca:

```
root# tail -1 /etc/group
juca:x:1002:
```

Para criar o usuário joca, no **grupo primário juca**, comandar:

```
root# /usr/sbin/useradd -m -s /bin/bash -g juca joca
```

No comando acima, a opção **-g** é o groupname do grupo onde será criado o usuário joca.

NOTA: para o comando acima funcionar, precisa existir o groupname juca.

Para excluir o usuário joca, comandar **userdel**:

```
root# userdel -r joca
```

No comando acima, a opção "**-r**" é para excluir a home do usuário.

Para criar um grupo, usar o comando **groupadd**:

```
root# groupadd jeca
```

Para excluir um grupo, usar o comando **groupdel**:

```
root# groupdel jeca
```

Para incluir o usuário juca também no grupo secundário aluno, usar o comando **usermod**:

```
root# groups juca
juca : juca mail
root# usermod -G mail,aluno juca
root# groups juca
juca : juca mail aluno
```

NOTA 1: o comando **groups** é mais simples que **id** para saber a que grupos o usuário pertence. No caso acima, juca inicialmente pertencia aos grupos primário juca e secundário mail. Depois de usar **usermod**, passou a pertencer também ao grupo secundário aluno.

NOTA 2: a sintaxe do **usermod** exige que se repita os grupos secundários aos quais o usuário pertence, senão eles serão excluídos.

12.2 - Qualidade da senha

Normalmente, o default dos sistemas Linux é nunca expirar a senha, deste modo o usuário nunca é obrigado a trocar a senha. Do ponto de vista da segurança, esta não é uma boa política.

Para saber o status da senha de um usuário, usar o comando **passwd** com opção -S:

```
root# passwd -S aluno
aluno PS 2017-02-15 0 99999 7 -1 (Definição de senha, criptografia SHA512.)
```

Na saída do comando acima, PS significa que o usuário não está bloqueado. Para bloquear (lock), usar o comando **passwd** com a opção **-l**:

```
root# passwd -l aluno
Bloqueando senha para o usuário aluno.
passwd: Sucesso
```

Agora, o status da senha vai mostra bloqueado (locked):

```
root# passwd -S aluno
aluno LK 2017-02-15 0 99999 7 -1 (Senha bloqueada.)
```

Na saída do comando acima, LK significa usuário bloqueado. Fisicamente, o bloqueio é obtido pela inserção do caractere "!" ou "*" no início do 2° campo do /etc/shadow (*hash de senha*). Isto pode ser verificado com o comando grep:

```
root# grep aluno /etc/shadow
aluno:!!$1$GXJzit5J$vS4wC8AW6hV8zvLu6Dtxc.:17213::::::
```

Para desbloquear o usuário, usar o comando **passwd** com a opção **-u** (unlock):

```
root# passwd  -u  aluno
Desbloqueando senha para o usuário aluno.
passwd: Sucesso
```

NOTA: bloquear ou desbloquear o usuário não implica em necessidade de alterar a senha.

Mas para forçar a senha a expirar periodicamente e obrigar o usuário a trocá-la regularmente, é necessário usar o comando passwd com a opção -x. Porém, quando a senha estiver próxima a expirar o usuário precisa ser informado disso, então devem ser usadas também as opções -n e -w. A sintaxe do comando com estas opções opções segue abaixo:

passwd [-n mindays][-x maxdays] [-w warndays]

Na sintaxe acima, as opções **-n**, **-x** e **-w** significam:

-n: número mínimo de dias sem poder alterar a senha depois de feita a troca;
-x: número de dias em que a senha é válida, depois disso é forçado a alterar de novo;
-w: número de dias antes da troca, quando começa a avisar que precisa alterar a senha.

Antes de impor a política para forçar a alterar a senha periodicamente, convém dar uma olhada nos campos 3, 4, 5 e 6 do arquivo /etc/shadow:

```
root# grep  aluno  /etc/shadow
aluno:$1$GXJzit5J$vS4wC8AW6hV8zvLu6Dtxc.:17213::::::
```

Notar que no 3° campo existe o número 17213, que é a data da última alteração de senha, e que também é o número de dias decorridos desde 1° de janeiro de 1970. Esta senha foi alterada pela última vez em 15 de fevereiro de 2017.

Hoje, 08 de setembro de 2017, é o dia de número 17417. Isto significa que, ao impor uma política de validade de senha por 30 dias, a senha de aluno ficará imediatamente expirada, pois:

17417 - 17213 > 30 (são decorridos mais de 30 dias desde a última alteração de senha)

Para impor esta política de alteração de senha, usar o comando **passwd** com as opções **-n**, **-x** e **-w**:

```
root# passwd  -n 2  -x 30  -w 8  aluno
Ajustando dados de expiração para o usuário aluno.
passwd: Sucesso
```

O status da senha agora vai mostrar:

```
root# passwd -S aluno
aluno PS 2017-02-15 2 30 8 -1 (Definição de senha, criptografia MD5.)
```

Uma inspeção de /etc/shadow vai mostrar:

```
root# grep aluno /etc/shadow
aluno:$1$GXJzit5J$vS4wC8AW6hV8zvLu6Dtxc.:17213:2:30:8:::
```

Na saída do comando acima, os campos 4, 5 e 6 contém agora os números 2, 30 e 8, que são os valores para a política de senha que foram impostos com as opções "-n 2 -x 30 -w 8" do comando passwd.

13 - UMASK

Umask é uma máscara usada na criação de arquivos. O valor que estiver configurado nessa máscara, que é individual, será usado como o modo de criação do arquivo.

Para saber qual a máscara, comandar:

```
shell$ umask
0002
```

Esse comando mostra que a máscara do usuário aluno é 0002. Então, se aluno criar um arquivo, as permissões serão **-rw-rw-r--**, que é o octal 0664. Se aluno criar um diretório, as permissões serão **drwxrwxr-x**, que é o octal 0775.

Esses valores de permissões na criação vêm da diferença binária entre 0666 para arquivos regulares e 0777 para diretórios.

Para calcular as permissões do arquivo criado, são usados os operadores bitwise NOT na máscara e posteriormente o bitwise AND binário entre 0666 e este número, para o caso de arquivo regular. Se fosse diretório, seria 0777.

Exemplos de operações bitwise NOT e AND:

- NOT(0) = 1
- NOT(1) = 0
- NOT(110) = 001
- 1 AND 0 = 0
- 1 AND 1 = 1
- 0 AND 0 = 0
- 001 AND 111 = 001

Como exemplo, calcular as permissões de um arquivo regular criado com **umask 0002**. Primeiramente, converter a máscara e o número 0666 para binário:

0002 = 000 000 000 010
0666 = 000 110 110 110

Depois, aplicar o operador NOT na máscara:

NOT(000 000 000 010) = 111 111 111 101

Por fim, calcular o operador AND destes números binários:

111 111 111 101 AND
000 110 110 110
---
000 110 110 100 => 0664 [**-rw-rw-r--**]

Num outro exemplo, usando essa mesma máscara, porém criando um diretório, é necessário primeiro converter 0777 para binário:

0777 = 000 111 111 111

Calcular o operador AND entre o NOT da máscara e 0777 binário:

111 111 111 101 AND
000 111 111 111
---
000 111 111 101 => 0775 [**drwxrwxr-x**]

Ainda como exemplo, alterar a máscara para 0027. Para isso, comandar umask e passar como parâmetro o novo valor:

```
shell$ umask 0027
shell$ umask
0027
```

Agora, calcular as permissões de arquivo regular e diretório criados com essa nova máscara.

Inicialmente, converte esta máscara para binário:

0027 = 000 000 010 111

Depois, aplicar o operador NOT nesta máscara:

NOT(000 000 010 111) = 111 111 101 000

Por fim, calcular o operador AND destes números binários para o caso de arquivo regular [0666]:

111 111 101 000 AND
000 110 110 110
---
000 110 100 000 => 0640 [-**rw-r-----**]

Para o caso de criação de diretório, usar 0777 [000 111 111 111]:

111 111 101 000 AND
000 111 111 111
---
000 111 101 000 => 0750 [**drwxr-x---**]

Ainda como exemplo, alterar a máscara para 0022. Para isso, comandar umask e passar como parâmetro o novo valor:

```
shell$ umask  0022
shell$ umask
0022
```

Agora, calcular as permissões de arquivo regular e diretório criados com esta nova máscara.

Inicialmente, converte esta máscara para binário:

0022 = 000 000 010 010

Depois, aplicar o operador NOT nesta máscara:

NOT(000 000 010 010) = 111 111 101 101

Por fim, calcular o operador AND destes números binários para o caso de arquivo regular [0666]:

111 111 101 101 AND

```
000 110 110 110
--------------------
000 110 100 100       =>      0644 [-rw-r--r--]
```

Para o caso de criação de diretório, usar 0777 [000 111 111 111]:

```
111 111 101 101 AND
000 111 111 111
--------------------
000 111 101 101       =>      0755 [drwxr-xr-x]
```

O comando *umask* com a opção **-S** apresenta a máscara na forma simbólica:

```
shell$ umask -S
u=rwx,g=rwx,o=rx
```

NOTA: à primeira vista, passa a impressão que é simples obter a máscara dadas as permissões, fazendo o caminho inverso do que foi feito acima. Ou seja, definir as permissões desejadas e depois calcular a máscara a ser adotada. No entanto, esse processo inverso não costuma dar certo pois o caminho não é unívoco. Por exemplo, para **arquivos regulares** as máscaras 0027 e 0026 dão o mesmo resultado, que são permissões 0640 [**-rw-r-----**], já para diretórios a máscara 0027 dá permissões 0750 [**drwxr-x---**], mas a máscara 0026 dá permissões 0751 [**drwxr-x--x**]. Por isso, na prática se usa máscara 0027 e não 0026.

Mas se for necessário calcular a máscara dadas as permissões, deverá usar o operador XOR:

```
0640   000 110 100 000                          <=    permissões desejadas
                        XOR
0666   000 110 110 110                          <=    máscara para arquivo regular
       --------------------
       000 000 010 110      =>    máscara: 0026


0750   000 111 101 000                          <=    permissões desejadas
                        XOR
0777   000 111 111 111                          <=    máscara para diretório
       --------------------
       000 000 010 111      =>    máscara: 0027
```

14 - COMANDOS ADICIONAIS

adduser: é linque simbólico para useradd, usado para incluir usuários no sistema local;
deluser: é linque simbólico para userdel, usado para excluir usuários do sistema local;
env: imprime todo o ambiente exportado. Também pode rodar um executável alterando o ambiente previamente à execução do comando;
printenv: imprime o ambiente total ou parte deste;
uname: mostra informações sobre o sistema;
hostname: mostra ou configura o nome da máquina ou domínio;
uniq: reporta ou omite linhas duplicadas: "uniq -d" mostra apenas linhas duplicadas, "uniq -u" mostra apenas linhas únicas (não duplicadas);
which: mostra o caminho completo para os comandos. Exemplo: "which passwd".

15 - PROCESSOS: background, prioridade e uso de recursos

A proposta deste tópico é entender um pouco mais a respeito dos processos.

15.1 - Processos em background

Quando se dispara um comando no shell, a execução normalmente é interativa, e então o processo assim gerado roda em **foreground** (primeiro plano). Neste caso, o processo está vinculado ao shell onde foi executado, e depende deste para continuar rodando. Por exemplo, se alguém dispara um comando e, antes de finalizar este comando fechar o shell, o processo filho é automaticamente finalizado também.

Mas um processo também pode rodar em **background** (plano de fundo ou segundo plano), e neste caso não depende do shell aonde foi iniciado o processo. Em background, o processo se torna independente do shell aonde foi executado.

Para entender como enviar um processo para background, será necessário criar o script abaixo, de nome sleep.sh, e dar permissão de execução:

```
shell$ vi  sleep.sh; chmod  755  sleep.sh
```

```
#!/bin/bash
echo "Script teste"
sleep 600
```

Na última linha deste script a entrada **sleep 600** vai deixar a aplicação em sleep (dormindo) por 600 segundos (10 minutos) após o disparo, ou seja, este processo não finaliza de imediato.

Com isso, em outro shell, este processo pode ser visto rodando com o comando **ps -ef**.

15.1.1 - Enviar para background com o caractere & no final da linha

Para executar o comando e o processo já ir para background, precisa incluir o caractere "&" no final da linha de execução deste script:

```
shell$ ./sleep.sh  >  saida.txt  2>&1  &
[1]  6912
```

Reparar que na execução acima, além do redirecionamento da saída padrão e de erro para o arquivo saida.txt, no final da linha tem um caractere **&**, que envia este processo para background.

Na saída do comando acima, o número 1 entre colchetes, [1], informa que este é o job de número 1 do shell. Para visualizar todos os jobs disparados do presente shell, comandar **jobs**:

```
shell$ jobs
[1]+  Executando          ./sleep.sh  >  saida.txt  2>&1  &
```

Para verificar se está mesmo rodando, usar o comando ps:

```
shell$ ps  -ef  |  grep  sleep.sh
aluno    6912  2021  0  19:53 pts/2    00:00:00  /bin/bash  ./sleep.sh
```

Se quiser parar este processo, tem duas alternativas: dar o comando **kill** no processo ou trazer de volta para foreground e depois enviar o sinal <CTRL>c.

15.1.2 - Parar o processo com kill

Uma vez determinado o PID do processo, usar o comando kill:

```
shell$ kill  -9  6912
shell$
[1]+  Morto               ./sleep.sh > saida.txt 2>&1
```

NOTA 1: no comando acima, o executável kill enviou o sinal -9 para o processo. Este sinal é o SIGKILL, usado para finalizar o processo.

NOTA 2: o processo em background não pode mais ser parado por <CTRL>c. Para pará-lo, é necessário dar o comando kill no PID do processo.

NOTA 3: para saber todos os sinais que o comando kill pode enviar, comandar **kill** com a opção **-l** (--list).

NOTA 4: caso houvesse mais de um processo de nome sleep.sh, todos eles poderiam ser mortos com o comando "**killall sleep.sh**". Ao contrário do comando kill, killall recebe como parâmetro o(s) nome(s) do(s) processo(s), e não o PID.

NOTA 5: o usuário sem privilégios de root só consegue matar seus próprios processos.

15.1.3 - Trazer o processo de volta para foreground

Para trazer de volta para o shell o processo que foi enviado para background, usar o comando **fg** com a opção número do job, precedido do caractere %. Mas como o processo foi morto anteriormente, agora precisa ser executado novamente:

```
shell$ ./sleep.sh  >  saida.txt  2>&1  &
[1]  7094
shell$ jobs
[1]+  Executando          ./sleep.sh > saida.txt 2>&1 &
shell$ fg  %1
./sleep.sh > saida.txt 2>&1
^C
shell$
```

Após o comando **fg %1**, acima, a execução do processo que estava em background retornou para foreground, e só então foi possível encerrar o processo com o sinal <CTRL>c.

15.1.4 - Enviar para background após iniciada a execução em foreground

Um outro caso é enviar a execução para background após ter sido iniciada a execução no shell sem uso do caractere & no final da linha. Neste caso, o processo está em foreground, e para enviá-lo para background é necessário o sinal <CTRL>z no shell:

```
shell$ ./sleep.sh  >  saida.txt  2>&1
^Z
[1]+  Parado              ./sleep.sh > saida.txt 2>&1
shell$
```

Neste caso, foi liberado o shell, porém o processo foi para background no status parado, situação em que não processa. O status pode ser verificado com o comando jobs:

```
shell$ jobs
[1]+  Parado              ./sleep.sh > saida.txt 2>&1
```

Repare que o processo não está executando. Nesta situação, existem dois caminhos a serem tomados: trazer a execução de volta para foreground ou mudar o status de parado para rodando.

Para trazer de volta para foreground, no mesmo shell aonde foi disparado o sinal <CTRL>z, precisa ser executado o comando fg. É o mesmo caso já apresentado acima.

15.1.5 - Alterar o status de parado para rodando em background

Neste caso, a intenção não é trazer de volta para foreground nem parar o processo, mas alterar o status de parado para rodando.

Isto é feito com o comando **bg**, passando como opção o número do job precedido do caractere %:

```
shell$ jobs
[1]+  Parado              ./sleep.sh  >  saida.txt  2>&1
shell$ bg  %1
[1]+  ./sleep.sh  >  saida.txt  2>&1  &
shell$ jobs
[1]+  Executando          ./sleep.sh  >  saida.txt  2>&1  &
```

Reparar que agora o processo voltou a executar em background.

15.2 - Prioridade de processos

Os processos têm prioridades de execução. Num Linux, vão de -20 a +19, sendo que -20 é a prioridade máxima. Ao disparar um executável, o usuário sem privilégios de root usa a prioridade default, que é 0 (zero). Depois de iniciada a execução, o usuário consegue diminuir a prioridade dos seus processos, nunca aumentar.

Os processos do sistema (processos de root) podem entrar na faixa abaixo de zero. Por exemplo, se for necessário usar SWAP, o sistema vai disparar um processo para disponibilizar esta memória adicional. E para executar esta tarefa, o processo do sistema precisa ter prioridade acima dos processos dos usuários comuns.

Para entender como funciona a prioridade do processo, serão necessários três usuários logados simultaneamente: aluno, que cria um script que consome muita CPU, juca, que também executa o script de aluno, e root, que monitora o consumo de CPU e altera a prioridade de um processo.

Inicialmente, aluno vai para o diretório /tmp e cria o script loop.sh:

```
aluno$ cd /tmp; vi loop.sh; chmod 755 loop.sh
```

```
#!/bin/bash
echo "Iniciando..."
while [ 2 -ge 1 ]; do
        echo "."
        sleep 2
done
```

Como 2 sempre será maior que 1, o script acima é um loop fechado, portanto sem condição de parada. Uma vez iniciado, não para mais. Tudo que ele faz é imprimir um ponto na tela a cada 2 segundos.

Aluno então dispara o seu script, e observa que ele fica enviando pontos para a tela:

```
aluno$ ./loop.sh
Iniciando…
.
.
.
```

Em outro shell, o root monitora com o comando top e não vê nenhum consumo excessivo de CPU.

Depois disso, aluno altera o script loop.sh, comenta a instrução "sleep 2" e dispara de novo:

```
aluno$ more  loop.sh
#!/bin/bash
echo  "Iniciando..."
while  [  2  -ge  1  ];  do
         echo  "."
         #sleep  2                  => esta linha agora está comentada e não é mais interpretada
done
aluno$ ./loop.sh
```

Em outro shell, o root monitora com top, e agora observa que o processo loop.sh está consumindo muita CPU:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 7963 aluno     20   0  113132   2672   2520 R  99,7  0,0   0:49.30 loop.sh
 7676 aluno     20   0 1151868 252412  88576 S   2,0  3,1   0:41.37 chrome
```

Noutro terminal, o usuário juca também o dispara o script loop.sh. Simultaneamente, o root permanece monitorando o desempenho da CPU e percebe que agora tem dois processos rodando: um de aluno e o outro do juca. Cada processo consome cerca de 50% da CPU:

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 7963 aluno     20   0  113132   2672   2520 R  49,4  0,0   0:49.30 loop.sh
 8257 juca      20   0  113132   2628   2480 R  48,7  0,0   0:15.16 loop.sh
```

Convém notar que os dois processos tem prioridade 0 (zero), conforme mostra o campo NI (nice) do top.

Para reduzir o consumo de CPU, o root baixa a prioridade do processo de juca, PID 8257, para o mínimo possível. Para isso, o root usa o comando **renice**, e depois volta a monitorar a CPU com o comando **top**:

```
root# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
root# renice  +19  -p  8257
8257 (process ID) old priority 0, new priority 19
root# top
```

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 7963 aluno     20   0  113132   2672   2520 R  93,4  0,0   0:49.30 loop.sh
 8257 juca      39  19  113132   2628   2480 R   5,2  0,0   0:15.16 loop.sh
```

Na saída do comando top, acima, agora o processo de juca (PID 8257), está com prioridade 19, conforme pode ser visto no campo NI.

E agora, o root percebe que o processo de aluno, PID 7963, passou a consumir mais de 90% da CPU. Isso é devido à baixa prioridade do processo de juca, que liberou a CPU para o processo de aluno usar.

Então, root decide parar o processo de aluno. Para isso usa o comando **kill**:

```
root# kill -9 7963
```

Novamente o usuário aluno dispara o seu script, porém agora da forma correta, usando o comando **nice**, para já iniciar este processo com baixa prioridade:

```
aluno$ nice -n 19 ./loop.sh
Iniciando…
.
.
.
```

O root monitora com **top** e percebe que agora os dois processos voltam a consumir, cada um, cerca de 50% da CPU. Só que agora, devido à baixa prioridade destes dois processos, eles não atrapalham os demais, pois estão usando apenas os recursos ociosos do sistema. Nesta situação, quando outro usuário executar alguma aplicação, ele terá por *default* prioridade maior que os dois scripts que estão rodando.

```
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 9921 aluno     39  19  113132   2672   2520 R  47,1  0,0   0:49.30 loop.sh
 8257 juca      39  19  113132   2628   2480 R  48,6  0,0   0:15.16 loop.sh
```

Um ponto importante a considerar, é que este exercício foi feito numa máquina virtual com um único processador (core). Caso a máquina tenha mais que um core, a carga de processos vai se distribuir entre todos os processadores, e o resultado será um pouco diferente do que foi obtido aqui.

15.3 - Processos e uso de recursos

No tópico anterior, prioridade de processos, o script usado consumia muito processador. Agora, o script vai consumir toda a memória RAM, depois todo o SWAP e por fim será finalizado pelo sistema. Enquanto estiver rodando, será monitorado o uso da memória com os comandos **free** e **top**.

Neste exemplo, o interpretador é o Perl, que tem uma sintaxe de programação diferente do shell script.

Para criar o perl script, usar o vi. Inicialmente, aluno vai para o diretório /tmp e lá cria o script mem.pl:

```
aluno$ cd  /tmp;  vi  mem.pl;  chmod  755  mem.pl
```

```
#!/usr/bin/perl
$cont=0;
for(;;){
        $cont++;
        $vetor[$cont]=$cont*10000;
}
```

Notar que de novo se trata de um loop fechado, pois este loop ***for*** não tem condição de parada. Este script não dá nenhuma saída, apenas consome memória.

Durante a execução de mem.pl, o vetor $vetor[$cont] vai aumentar o número de elementos (índices) à medida que o loop correr, e os valores atribuídos a este vetor vão ocupar mais e mais memória.

O usuário aluno dispara esse script num terminal e o root monitora em outro.

```
aluno$ cat  mem.pl
#!/usr/bin/perl
$cont=0;
for(;;){
        $cont++;
        $vetor[$cont]=$cont*10000;
}
aluno$ ./mem.pl
```

Alternando entre os comandos **top** e **free**, o root observa que esse script rapidamente aloca

toda a memória disponível:

```
root# top
```

16 - COMANDOS DE REDE

Os comandos a seguir são usados para administrar a rede num Linux.

Para mostrar o dispositivo de rede [hardware], o comando é **lspci**:

```
shell$ lspci | grep -i ethernet
00:0f.0 Ethernet controller: nVidia Corporation MCP73 Ethernet (rev a2)
```

Para mostrar a configuração lógica da(s) interface(s) de rede(s), o comando é **ifconfig**:

```
shell$ ifconfig
eth0   Link encap:Ethernet  Endereço de HW 00:21:97:80:7C:FF
       inet end.: 192.168.1.10  Bcast:192.168.1.255  Masc:255.255.255.0
       endereço inet6: fe80::221:97ff:fe80:7cff/64 Escopo:Link
       UP BROADCASTRUNNING MULTICAST  MTU:1500  Métrica:1
       RX packets:55602 errors:0 dropped:0 overruns:0 frame:0
       TX packets:10620 errors:0 dropped:0 overruns:0 carrier:0
       colisões:0 txqueuelen:1000
       RX bytes:8883790 (8.4 MiB)  TX bytes:2304141 (2.1 MiB)
       IRQ:28 Endereço de E/S:0x6000

lo     Link encap:Loopback Local
       inet end.: 127.0.0.1  Masc:255.0.0.0
       endereço inet6: ::1/128 Escopo:Máquina
       UP LOOPBACKRUNNING  MTU:16436  Métrica:1
       RX packets:18 errors:0 dropped:0 overruns:0 frame:0
       TX packets:18 errors:0 dropped:0 overruns:0 carrier:0
       colisões:0 txqueuelen:0
       RX bytes:1268 (1.2 KiB)  TX bytes:1268 (1.2 KiB)
```

Ao invés do ifconfig, poderia ser usado o comando **ip**:

```
shell$ ip  show  addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP qlen 1000
   link/ether 00:21:97:80:7C:FF brd ff:ff:ff:ff:ff:ff
   inet 192.168.1.10/24 brd 192.168.1.255 scope global dynamic eth0
      valid_lft 589809sec preferred_lft 589809sec
   inet6 fe80::221:97ff:fe80:7cff/64 scope link
      valid_lft forever preferred_lft forever
```

Para mostrar as configurações da interface de rede [como root], o comando é **ethtool**:

```
root# ethtool  eth0
Settings for eth0:
      Supported ports: [ MII ]
      Supported link modes:   10baseT/Half 10baseT/Full
                              100baseT/Half 100baseT/Full
                              1000baseT/Full
      Supports auto-negotiation: Yes
      Advertised link modes:  10baseT/Half 10baseT/Full
                              100baseT/Half 100baseT/Full
      Advertised auto-negotiation: Yes
      Speed: 100Mb/s
      Duplex: Full
      Port: MII
      PHYAD: 1
      Transceiver: external
      Auto-negotiation: on
      Supports Wake-on: g
      Wake-on: d
      Link detected: yes
```

Para mostrar o conteúdo da tabela de rotas, o comando é **route**:

```
shell$ route  -n
Tabela de Roteamento IP do Kernel
Destino         Roteador        MáscaraGen.     Opções Métrica  Ref   Uso  Iface
192.168.1.0     0.0.0.0         255.255.255.0   U      1        0     0    eth0
0.0.0.0         192.168.1.1     0.0.0.0         UG     0        0     0    eth0
```

NOTA: na saída do comando acima, a opção UG indica que a rota usa um roteador. No caso acima, trata-se da rota padrão (default gateway).

Para mostrar o conteúdo da tabela de rotas, também poderia ser usado o comando **ip**:

```
shell$ ip  route  show
default via 192.168.1.1 dev eth0  proto static  metric 425
192.168.1.0/24 dev eth0  proto kernel  scope link  src 192.168.1.10  metric 425
```

Os comandos, **ifconfig**, **ethtool**, **route** e **ip** também são usados para reconfigurar a rede. Por exemplo:

```
root# ifconfig  eth0  192.168.1.10  netmask 255.255.255.0  up
```

O comando acima configura o endereço IP 192.168.1.10/255.255.255.0 na interface eth0.

Se o sistema Linux fosse um CentoOS [que é baseado no Red Hat], para esta configuração resistir a um boot precisaria ser incluída no arquivo **/etc/sysconfig/network-scripts/ifcfg-eth0**. Caso não fosse a interface **eth0**, poderia ser **eth1**, ou **eth2**, etc, então os arquivos seriam **ifcfg-eth1**, ou **ifcfg-eth2**, etc.

Se fosse um sistema Ubuntu [baseado no Debian], para esta configuração resistir a um boot precisaria ser incluída no arquivo **/etc/network/interfaces**.

Com o comando ip, para configurar o endereço IP 192.168.1.10 com máscara /24, seria:

```
root# ip  addr  add  192.168.1.10/24  dev  eth0
```

Para configurar a rota default gateway 192.168.1.1 para a interface **eth0**, usar o comando **route**:

```
root# route  add  default  gw  192.168.1.1  eth0
```

Num CentOS, para esta rota resistir a um boot esta precisaria ser configurada no arquivo **/etc/sysconfig/network-scripts/route-eth0**, isso em caso de ser usada a interface **eth0**.

Para saber se o equipamento está acessando a rede, usar o comando **ping**. Por exemplo:

```
shell$ ping  localhost
PING localhost (127.0.0.1) 56(84) bytes of data.
64 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.076 ms
64 bytes from localhost (127.0.0.1): icmp_seq=2 ttl=64 time=0.045 ms
64 bytes from localhost (127.0.0.1): icmp_seq=3 ttl=64 time=0.041 ms
64 bytes from localhost (127.0.0.1): icmp_seq=4 ttl=64 time=0.039 ms
^C
--- localhost ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3308ms
rtt min/avg/max/mdev = 0.039/0.050/0.076/0.015 ms
```

```
shell$ ping  192.168.1.20
 ping 192.168.1.20
PING 192.168.1.20 (192.168.1.20) 56(84) bytes of data.
64 bytes from 192.168.1.20: icmp_seq=1 ttl=64 time=38.6 ms
64 bytes from 192.168.1.20: icmp_seq=2 ttl=64 time=9.33 ms
64 bytes from 192.168.1.20: icmp_seq=3 ttl=64 time=12.0 ms
64 bytes from 192.168.1.20: icmp_seq=4 ttl=64 time=10.3 ms
64 bytes from 192.168.1.20: icmp_seq=5 ttl=64 time=19.1 ms
^C
--- 192.168.1.20 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4846ms
rtt min/avg/max/mdev = 9.337/17.902/38.698/10.948 ms
```

Para traçar uma rota até o equipamento, usar o comando **traceroute** [no DOS/Windows é tracert]. Por exemplo:

```
root# traceroute  -n  104.28.0.37
```

No comando acima, a opção **-n** é para não resolver os nomes dos equipamentos, que agiliza o trace por não usar o serviço DNS.

Para resolver o IP de determinado host, usar o comando **nslookup**, **dig** ou **host**:

```
shell$ nslookup  www.jairo.pro.br
Server:         186.251.39.121
Address:        186.251.39.121#53

Non-authoritative answer:
www.jairo.pro.br        canonical name = www.jairo.pro.br.cdn.cloudflare.net.
Name:   www.jairo.pro.br.cdn.cloudflare.net
Address: 104.28.0.37
Name:   www.jairo.pro.br.cdn.cloudflare.net
Address: 104.28.1.37
```

No caso do comando nslookup, acima, o serviço de nomes que fez a resolução do nome jairo.pro.br, e obteve o endereço IP, foi 186.251.39.121. O serviço de nomes no endereço IP 186.251.39.121 foi consultado por estar declarado no arquivo **/etc/resolv.conf**:

```
shell$ more  /etc/resolv.conf
search  uninove.br
nameserver  186.251.39.121
nameserver  186.251.39.122
```

NOTA 1: por default, é consultado o primeiro serviço de nomes declarado no arquivo **/etc/resolv.conf**. Mas se este estiver inacessível, é tentando o segundo, e assim por diante para o caso de haver mais de dois endereços IP declarados.

NOTA 2: o arquivo **/etc/resolv.conf** é para configuração de cliente de serviço de nomes, e não para serviço de nomes.

O comando **dig** faz a mesma coisa que o nslookup, e tem a vantagem de mostrar a demora da consulta (query time):

```
shell$ dig  www.jairo.pro.br

; <<>> DiG 9.9.4-RedHat-9.9.4-38.el7_3 <<>> www.jairo.pro.br
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24137
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;www.jairo.pro.br.              IN      A

;; ANSWER SECTION:
www.jairo.pro.br.       3486    IN      CNAME   www.jairo.pro.br.cdn.cloudflare.net.
www.jairo.pro.br.cdn.cloudflare.net. 186 IN A   104.28.0.37
www.jairo.pro.br.cdn.cloudflare.net. 186 IN A   104.28.1.37

;; Query time: 19 msec
;; SERVER: 186.251.39.121#53(186.251.39.121)
;; WHEN: Sat Fev 4 15:47:22 BRST 2017
;; MSG SIZE  rcvd: 126
```

Para a resolução de nomes, o comando **host** é o mais simples de todos, pois mostra apenas o endereço IP:

```
shell$ host  www.jairo.pro.br
www.jairo.pro.br is an alias for www.jairo.pro.br.cdn.cloudflare.net.
www.jairo.pro.br.cdn.cloudflare.net has address 104.28.1.37
www.jairo.pro.br.cdn.cloudflare.net has address 104.28.0.37
www.jairo.pro.br.cdn.cloudflare.net has IPv6 address 2400:cb00:2048:1::681c:25
www.jairo.pro.br.cdn.cloudflare.net has IPv6 address 2400:cb00:2048:1::681c:125
```

Para ver as conexões de rede, tabelas de roteamento e estatísticas de interface, usar o comando **netstat**:

```
shell$ netstat -na | head
Conexões Internet Ativas (servidores e estabelecidas)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               OUÇA
tcp        0      0 127.0.0.1:631           0.0.0.0:*               OUÇA
tcp        0      0 192.168.1.10:47311      64.233.163.19:80        ESTABELECIDA
tcp        0      0 192.168.1.10:47312      64.233.163.19:80        ESTABELECIDA
tcp        0      0 ::1:631                 :::*                    OUÇA
udp        0      0 0.0.0.0:47061           0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 0.0.0.0:631             0.0.0.0:*
```

O comando **netstat -nr** também mostra a tabela de rotas:

```
shell$ netstat -nr
Tabela de Roteamento IP do Kernel
Destino     Roteador    MáscaraGen.     Opções Métrica Ref  Uso Iface
192.168.1.0 0.0.0.0     255.255.255.0   U      1       0    0   eth0
0.0.0.0     192.168.1.1 0.0.0.0         UG     0       0    0   eth0
```

Com a opção **tp** é possível saber o PID [Process IDentifier] do processo que abriu a porta. A opção **t** indica o PID e **p** a porta:

```
shell$ netstat -tpn
Conexões Internet Ativas (sem os servidores)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 192.168.1.10:49657      187.73.33.34:80         TIME_WAIT   -
tcp        0      0 192.168.1.10:56699      74.125.234.54:443       ESTABELECIDA2364/firefox-bin

shell$ ps -p 2364
PID  TTY      TIME     CMD
2364 pts/0    00:07:22 firefox-bin
```

O comando **netstat -ns** mostra as estatísticas de pacotes trafegados na rede.

A associação entre nomes de protocolos e suas respectivas portas TCP estão nos arquivos **/etc/services** e **/etc/protocols**.

Para descobrir quais serviços determinado host disponibiliza, use o comando **nmap**, que faz um scan de portas:

```
root# nmap  localhost
Starting Nmap 4.76 ( http://nmap.org ) at 2009-08-23 17:02 BRT
Warning: Hostname localhost resolves to 2 IPs. Using 127.0.0.1.
Interesting ports on localhost (127.0.0.1):
Not shown: 998 closed ports
PORT        STATE       SERVICE
22/tcp      open        ssh
631/tcp     open        ipp
```

```
root# nmap  192.168.1.20
Starting Nmap 4.76 ( http://nmap.org ) at 2009-08-23 17:02 BRT
Warning: Hostname localhost resolves to 2 IPs. Using 127.0.0.1.
Interesting ports on localhost (127.0.0.1):
Not shown: 998 closed ports
PORT        STATE       SERVICE
22/tcp      open        ssh
25/tcp      open        smtp
80/tcp      open        http
3128/tcp    open        squid-http
```

O cliente *wget* acessa o serviço web em linha de comando, e faz *dowload* de arquivos.

Por exemplo, para acessar o serviço web no endereço 192.168.1.10, o comando é:

```
shell$ cd  /tmp
shell$ wget  192.168.1.10
--2013-06-27 20:29:57--  http://192.168.1.10/
Conectando-se a 192.168.1.10:80... conectado.
A requisição HTTP foi enviada, aguardando resposta... 200 OK
Tamanho: 726 [text/html]
Salvando em: “index.html”
100%[======================================>] 726         --.-K/s   em 0s
2013-06-27 20:29:57 (91,9 MB/s) - “index.html” salvo [726/726]
```

No caso, antes de dar o comando *wget* foi usado o comando *cd*, isto para baixar o arquivo index.html de 192.168.1.10 para o diretório local /tmp.

Caso haja um proxy no meio do caminho, que é o caso de acesso à internet a partir dos laboratórios acadêmicos da Uninove, precisa antes passar a instrução de usuário e senha para o wget, e isso é feito com o comando **export**:

```
shell$ export  http_proxy=http://RA:SENHA@186.251.39.92:3128
```

Onde:

RA:	é o RA do aluno;
SENHA:	é a senha de acesso do aluno;
186.251.39.92:	é o IP do serviço proxy, que atende na porta **3128** (é um Squid).

17 - GERENCIAMENTO DE PACOTES

Nas distribuições Linux, pacote é um arquivo contendo um conjunto de arquivos, que podem ser executáveis, configurações, documentos, bibliotecas, etc. A função do pacote é garantir uma maneira prática de instalar, remover e atualizar o sistema operacional e suas aplicações.

Usualmente, quem cria e mantém esses pacotes é a própria distribuição Linux. Normalmente, a mídia de instalação do Linux contém pacotes a serem instalados.

Para gerenciar pacotes precisa ser root. O jeito simples de adquirir privilégios de root é com o comando sudo. Por exemplo:

```
shell$ id
uid=1000(aluno) gid=1000(aluno) grupos=1000(aluno),3(sys),12(mail)
shell$ sudo  su  -
[sudo] password for aluno:
root# id
uid=0(root) gid=0(root) grupos=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

Após enviar a senha de aluno, o usuário ganha o shell de root, como pode ser verificado com o símbolo do shell, que agora é uma cerquilha "#".

17.1 - Gerenciamento no Linux CentOS

CentOS significa *Community ENTerprise Operating System*. Na prática, o CentOS se alimenta do código fonte disponibilizado pela Red Hat: toda vez que a Red Hat lança uma nova

versão, o seu código fonte é "reempacotado" com pequenas modificações para criar uma nova versão do CentOS. A lógica por trás dessa cópia é que o CentOS permite instalação e atualização do sistema e aplicações mesmo sem pagamento de licença de uso.

De um modo geral, os pacotes de instalação e atualização do Red Hat são os mesmos do CentOS, propositalmente mantidos compatíveis.

Os comandos para gerenciar pacotes no Red Hat são **rpm** e **yum**. O comando rpm é para gerenciar pacotes localmente, e o yum pela rede.

Como exemplo de utilização dos comandos ***rpm*** e ***yum***, será utilizado o pacote rpm *nmap-5.51-1.i386.rpm*. Reparar que o pacote é um arquivo.

17.1.1 - rpm

Para checar se o pacote já está instalado, comandar:

```
shell$ rpm -aq | grep nmap
```

No comando acima, se houvesse algum pacote instalado que contivesse a palavra *nmap*, seria mostrado.

Depois de baixar o arquivo *nmap-5.51-1.i386.rpm*, para saber quais arquivos este pacote contém, comandar:

```
shell$ rpm -qlp nmap-5.51-1.i386.rpm
/usr/bin/ndiff
/usr/bin/nmap
/usr/share/doc/nmap-5.51
/usr/share/doc/nmap-5.51/COPYING
/usr/share/doc/nmap-5.51/README
/usr/share/doc/nmap-5.51/nmap.usage.txt
/usr/share/man/de/man1/nmap.1.gz
...
```

Para instalar um pacote rpm, comandar:

```
root# rpm -ivh nmap-5.51-1.i386.rpm
Preparando                ########################################### [100%]
   1:nmap                 ########################################### [100%]
```

Para saber à qual pacote determinado arquivo pertence, usar a opção **-qf**:

```
root# which  nmap
/usr/bin/nmap
root# rpm  -qf  /usr/bin/nmap
nmap-5.51-1.i386.rpm
```

Para excluir um pacote rpm que já esteja instalado, comandar:

```
root# rpm  -e  nmap-5.51-1.i386
```

Para atualizar um pacote que já esteja instalado, comandar:

```
root# rpm  -Uvh  nmap-5.51-1.i386
```

17.1.2 - yum

O *yum* faz o mesmo gerenciamento de pacotes que o comando *rpm*, o diferencial é que o *yum* busca pelo[s] pacote[s] necessário[s] para a instalação em algum repositório de pacotes na rede, que também pode ser na internet. Mais interessante, alguns pacotes, para serem instalados, necessitam de pré-requisitos, e nesse caso o próprio *yum* faz os downloads e instalações sozinho. Se fosse o comando *rpm*, ele pararia a instalação e avisaria que está faltando algum pacote que é pré-requisito para aquela instalação.

Para instalar algum pacote com o yum, comandar:

```
root# yum  install  nmap
Plugins carregados: refresh-packagekit
---> Pacote  nmap-5.51-1.i386.rpm  definido para ser atualizado
Tamanho total do download: 2.4 M
Correto? [s/N]:s
Baixando pacotes:
nmap-5.51-1.i386.rpm                                    | 2.4 MB     00:20
Executando o rpm_check_debug
  Instalando     : 2: nmap-5.51-1.i386.rpm                                  1/1

Instalados:
  nmap-5.51-1.i386.rpm
```

O *yum* também poderia ser usado para atualizar um pacote. Por exemplo:

```
root# yum update nmap
```

Mas para que o *yum* funcione a contento, precisa ser definido quais são os endereços de repositórios de pacotes na internet, que serão usados para download. Essa configuração está no arquivo "**/etc/yum.conf**". Normalmente, este arquivo não precisa ser configurado, pois se for necessário adicionar um novo site repositório basta incluir um arquivo de configuração adicional no diretório "**/etc/yum.repos.d**".

Por exemplo, o arquivo "**/etc/yum.repos.d/CentOS-Base.repo**" contém, entre outras coisas, o seguinte conteúdo:

```
root# head /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5
protect=1

#released updates
[updates]
```

17.2 - Gerenciamento no Linux Ubuntu

A distribuição Ubuntu é derivada do Debian, portanto o gerenciamento de pacotes locais é feito com o comando **dpkg**, e pela rede é feito com o comando **apt-get**.

Para saber se o pacote **vim** está instalado, usar o comando **dpkg** com a opção **-l**:

```
root# dpkg -l | grep -i vim
ii  vim              2:7.4.1689-3ubuntu1.2   amd64        Vi IMproved - enhanced vi editor
ii  vim-common       2:7.4.1689-3ubuntu1.2   amd64        Vi IMproved - Common files
ii  vim-runtime      2:7.4.1689-3ubuntu1.2   all          Vi IMproved - Runtime files
```

NOTA: para instalar um pacote de nome vim.deb, usar o comando "**dpkg -i vim.deb**".

O comando **apt-get** lê o arquivo de configuração **/etc/apt/sources.list** para determinar os locais com repositório de pacotes para instalação.

Normalmente, a configuração em **sources.list** está correta, porém com frequência dá erro na instalação pois aparece na primeira linha uma configuração do tipo:

```
root# more /etc/apt/sources.list
...
deb cdrom:[Debian GNU/Linux 5.0.1 _Lenny_ - Official i386 DVD Binary-1 20090413-00:33]/ lenny
contrib main
...
```

Se houver a linha acima, isto indica que **apt-get** está configurado para buscar pacotes na mídia de cd-rom, e não num repositório de pacotes na rede.

Se este for o caso, a solução então é editar o arquivo **sources.list** e comentar essa linha [colocar uma cerquilha "#" na frente da linha]:

 # deb cdrom:[Debian GNU/Linux 5.0.1 _Lenny_ - Official i386 DVD Binary-1 20090413-00:33]/ lenny contrib main

Depois disso é necessário atualizar o database de repositórios com o comando **apt-get update**:

```
root# apt-get update
```

Depois disso, o comando para instalação de pacotes deverá funcionar. Por exemplo, para instalar o pacote **nmap**, comandar:

```
root# apt-get install nmap
```

O comando **apt-get** normalmente instala pacotes disponibilizados em repositórios na internet, por isso é muito prático.

Porém, pode ser o caso de não haver acesso à internet, então a aplicação poderia ser instalada diretamente de um pacote "**.deb**" no CD-ROM.

Para usar o CD-ROM [ou DVD] como local de repositório de pacote, usar o comando apt-cdrom para declaração o CD-ROM como repositório:

```
root# apt-cdrom  add
```

Também poderia ser o caso de querer instalar o pacote diretamente de um arquivo "**.deb**", então neste caso poderia ser usado o comando **dpkg**:

```
root# dpkg  -i  nmap-4.76.deb
```

No comando acima, a opção **-i** é para instalar o pacote nmap, do arquivo local nmap-4.76.deb. Para remover o pacote, usar a opção **-r**.

Para saber se determinada aplicação está disponível para instalação via apt, usar o comando **apt-cache** para descobrir:

```
root# apt-cache  search  nmap
```

O comando acima mostra os pacotes disponíveis que contêm o nome nmap.

O comando **apt-key** serve para gerenciar uma lista de chaves usadas pelo apt para autenticar pacotes. Pacotes autenticados por essas chaves são considerados confiáveis.

Existe também o **aptitude**, que é um comando para instalar e remover pacotes pela rede, semelhante ao apt-get. O diferencial é que aptitude se apresenta melhor que o apt-get no tópico dependências de instalação de pacotes.

17.3 - Gerenciamento no Unix em geral

O gerenciamento de pacotes usa diferentes comandos para os diferentes sistemas operacionais da família Unix.

No Solaris 10 (e anteriores) usa-se **pkgadd** e **pkgrm** para adicionar e remover pacotes. Para obter informações sobre pacotes instalados, é usado o comando **pkginfo**. Já no Solaris 11 (atual em 2017), o comando é **pkg** tanto para instalar ou remover pacotes.

No AIX também tem o gerenciador de pacotes **rpm**, igual ao do Linux. Mas neste sistema é bastante comum usar o comando **smitty** [ou **smit**], que é a ferramenta padrão de administração/configuração do sistema.

Por exemplo, para instalar, entrar no diretório onde está o pacote e comandar:

```
root# smitty  install
```

Uma vez disparado o comando acima, ele vai oferecer um menu com seleção de *file sets* para instalar.

18 - ANEXOS

18.1 - EUID/EGID

O código fonte em C, abaixo, pode ser compilado com o gcc (GNU project C and C++ compiler), e o executável gerado demonstra o conceito de EUID e EGID.

O código do programa **uid-euid.c** é:

```
===uid-euid.c=============================================================
#include <stdio.h>
#include <unistd.h>

int main (void) {
        printf("uid: %d\neuid: %d\ngid: %d\negid: %d\n", getuid(), geteuid(), getgid(), getegid());
}
==========================================================================
```

Para compilar o código acima, o primeiro passo é copiar e colar para um arquivo de texto chamado uid-euid.c.

A seguir, o usuário aluno compila no **/tmp** o programa uid-euid.c com o comando:

"**gcc -o uid-euid uid-euid.c**".

Isso vai gerar o executável **uid-euid**.

Na sequência, o usuário aluno, UID/GID 1000/1000, dá a permissão SUID com o comando:

chmod 4755 /tmp/uid-euid

Posteriormente, outro usuário (por exemplo juca, UID/GID 1001/1001), comanda o executável **/tmp/uid-euid**. O resultado no seu shell será:

```
----------------
uid: 1001
euid: 1000
```

```
gid: 1001
egid: 1001
----------------
```

O raciocínio feito para demonstrar o EUID é o mesmo para EGID. O usuário aluno comanda:

chmod 2755 /tmp/uid-euid

Após, o usuário juca comanda de novo **/tmp/uid-euid** e recebe como saída:

```
----------------
uid: 1001
euid: 1001
gid: 1001
egid: 1000
----------------
```

Agora o usuário aluno comanda:

chmod 6755 /tmp/uid-euid

O usuário juca agora recebe como saída:

```
----------------
uid: 1001
euid: 1000
gid: 1001
egid: 1000
----------------
```

18.2 - Script simples de backup

No exemplo abaixo é apresentado um script simples de backup.

```
--- backup.sh ----------------------------------------
#!/bin/bash
# backup diario de  /bin
if [ -s backup.tar.gz ]; then
        mv  -f  backup.tar.gz  backup.tar.gz.OLD
fi
tar  -cvf backup.tar  /bin
```

```
gzip  backup.tar
data_agora=`date`
string="Backup finalizado em  $data_agora"
echo  $string
------------------------------------------------------------
```