Introdução à Utilização de Sistemas Unix

Dulce Domingos e Hugo Miranda

Março/2015

Este documento é baseado nos seguintes:

- Hugo Miranda, "Introdução à Utilização de Sistemas Unix", versão 1.1. Departamento de Informática. Fevereiro de 2003.
- Henrique João L. Domingos, Teresa Chambel, "O Shell do Sistema Unix - utilização, filtros e programação", folhas de apoio às aulas práticas de Sistemas de Exploração I, DI-FCUL, 1991.
- Brian W. Kernighan, Rob Pike, "The Unix Programming Environment", Prentice Hall Inc, 1978.

Índice

1. Introdução

O Unix é um sistema operativo criado no início da década de 70 nos Bell Lab's nos Estados Unidos. É um sistema multi-utilizador desenvolvido na linguagem de programação C. Actualmente existem diversas versões (por exemplo, BSD Unix, System V, HPUX, Solaris).

O Linux é uma versão do Unix para pequenos computadores desenvolvida por Linus Torvalds. Para além das possibilidades que oferece, o Linux tem-se expandido mundialmente por ser de distribuição gratuita. Actualmente, o Linux é mantido por um conjunto de programadores que de forma gratuita o vão actualizando. Presentemente existem diversas versões disponíveis. A opção por uma prende-se normalmente com razões culturais embora algumas apresentem vantagens sobre as restantes.

Um dos grandes distribuidores de software para Linux é a FSF (Free Software Foundation) com o seu projecto GNU. São deles as mais populares versões de compiladores (gcc), debuggers (gdb), editores de texto (gnu emacs) e utilitários de compressão (gzip).

Na prática, qualquer pessoa pode contribuir para o desenvolvimento do Linux produzindo e distribuindo software. Grandes outras marcas têm disponibilizado software de forma gratuita para este sistema operativo.

2. Ambiente dos laboratórios

Todos os computadores dos laboratórios têm instalado o sistema operativo Linux.

O Linux disponibiliza sessões independentes em simultâneo. Por sessão entende-se um ambiente de trabalho completamente independente dos restantes onde o utilizador tem que se registar e sair. As sessões podem ser comutadas entre si pela utilização de Alt+Fn onde n é o número da sessão pretendida. O ambiente gráfico X Windows ocupa normalmente a sessão 7 ou a sessão 1. Para obter uma sessão sem este ambiente gráfico (designada *sessão em modo de texto*), basta seleccionar qualquer uma das restantes. Por exemplo, a sessão 1 é obtida por Ctrl+Alt+F1.

Os computadores dos laboratórios têm instalado um sistema de ficheiros distribuído denominado NFS (Network File System) que lhes permite aceder aos ficheiros guardados na

máquina onde foram criadas as áreas da disciplina. Na eventualidade de esta máquina não se encontrar disponível, todos os ficheiros ficarão inacessíveis.

A entrada numa sessão em Linux é feita pela digitação de um par (*login*, *password*). O *login* é atribuído pelo centro de informática e é permanente (do tipo fcXXXXX). A *password* é inicialmente atribuída pelo Centro de Informática e deverá ser alterada com regularidade.

A saída de uma sessão em modo texto será feita pelo comando logout, exit ou ainda, nalgumas configurações por ctrl^D. Após a terminação da sessão surgirá novamente o diálogo de abertura de sessão. As sessões gráficas dispõem de comandos para o efeito.

Atenção: É frequente, ao longo de um dia de trabalho, os utilizadores abrirem várias sessões na mesma máquina. Se for esse o caso, assegure-se do encerramento de todas elas percorrendo-as (Alt+F1 a Alt+F7) antes de abandonar a máquina.

Os ficheiros dos grupos das disciplinas estão na directoria **areas_de_grupo**.

2.1. Alteração da *password*

Uma boa *password* deve respeitar algumas regras, das quais destacamos:

- Conter pelo menos 8 caracteres
- Não formar uma palavra nem uma referência facilmente associável ao utilizador (por exemplo, nome próprio ou *login*)
- Incluir uma mistura de letras (preferencialmente maiúsculas e minúsculas), algarismos e símbolos.

Os utilizadores dos laboratórios do DI alteram a password na página do CI. O comando `password` dos sistemas Unix não permite alterar a *password* dos utilizadores dos laboratórios do DI.

3. Ficheiros e directorias

Em Unix, o nome de um ficheiro ou de uma directoria pode ter até 256 caracteres. Podem ser utilizados quaisquer caracteres excepto a barra (“/”), embora seja conveniente restringir essa utilização aos caracteres alfanuméricos (letras e algarismos) e aos caracteres “_” (underscore) e “.” (ponto).

3.1. Convenções de designação dos ficheiros

É vulgar encontrar em Unix o nome dos ficheiros dividido em duas partes pelo caracter “.”. A segunda parte identifica a *extensão* do ficheiro.

Alguns comandos do Unix são restritivos quanto às extensões que esperam nos ficheiros com que lidam. Os compiladores são disso um bom exemplo. Assim, qualquer programa escrito na linguagem C deverá ter a extensão “c” para ser compilado, enquanto que, por exemplo, em java a extensão é “java”.

Notas:

- Ao contrário do MS-DOS, o sistema operativo Unix não faz interpretação das extensões dos ficheiros. Um ficheiro é, aos olhos do sistema operativo, executável se o utilizador detiver **permissões** para tal e não se contiver uma dada extensão. Tipicamente, **no UNIX um ficheiro executável não possui extensão.**
- Em algumas situações é usual o nome do ficheiro conter vários caracteres “.”. A utilização do caracter “.” não sofre de qualquer restrição, podendo ocorrer diversas vezes num mesmo ficheiro. O número de caracteres após este também não é limitado.
- Alguns ficheiros especiais começam com o “.”. É o caso de um ficheiro associado a cada área de trabalho em Unix (`.bash_profile`), cujo conteúdo é executado sempre

que é efectuada uma entrada nessa área. Este tipo de ficheiros tem a particularidade de não ser visível na listagem de ficheiros de uma directoria (comando `ls`) excepto quando é utilizada a opção `-a`.

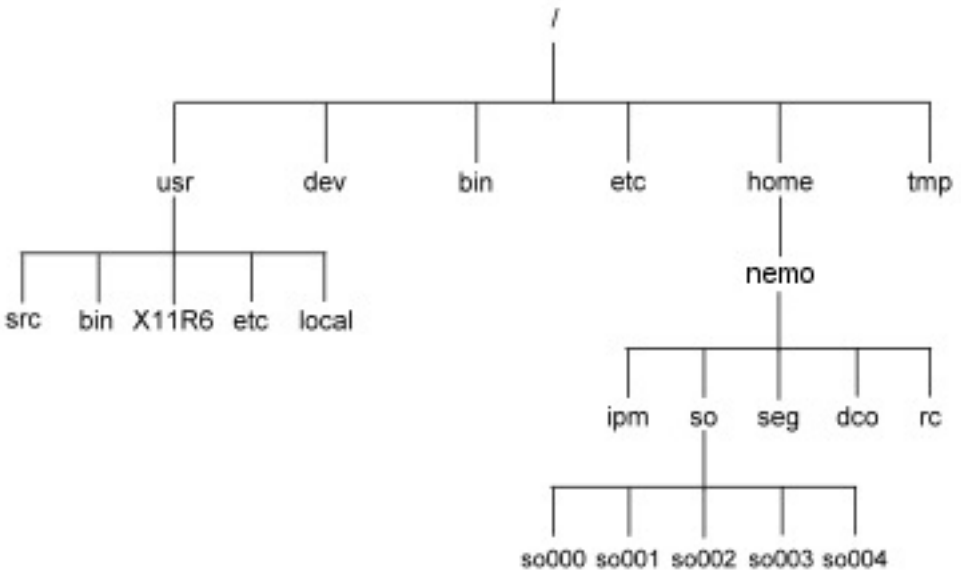
3.2. Estrutura

As directorias constituem um tipo especial de ficheiros. A sua função é agrupar ficheiros e outras directorias. Estes dir-se-ão *contidos* naquela.

Uma *subdirectoria* é um abuso de linguagem utilizado para designar uma directoria que se encontra dentro de uma outra.

No que se segue, as características de uma directoria, enquanto ficheiro Unix, serão esquecidas, sendo alvo da nossa atenção apenas o conceito (lógico) de directoria que será necessário para o utilizador do sistema.

Todos os ficheiros no Unix são guardados logicamente segundo uma estrutura hierárquica em árvore.



Notas:

- A árvore tem uma única *raiz*, identificada por "/". Este é o *nome* da raiz de qualquer sistema de ficheiros Unix. Todos os ficheiros e directorias são colocados abaixo da raiz.
- Existem duas directorias chamadas `etc`. Dois ficheiros (regulares ou directorias) podem ter o mesmo nome desde que não se encontrem dentro da mesma directoria.

Os ficheiros e as directorias podem ser identificados através de nomes absolutos ou através de nomes relativos.

Os nomes absolutos são construídos através da concatenação dos nomes das várias directorias desde a directoria raiz. O separador entre nomes é também a barra (/). Por exemplo, a directoria `so000` tem o seguinte nome absoluto:

```
/home/nemo/so/so000
```

enquanto que o nome absoluto da directoria `local` é:

```
/usr/local
```

Os nomes relativos são utilizados para identificar ficheiros ou directorias a partir de uma determinada directoria:

- Para referir uma subdirectoria da directoria corrente utiliza-se o nome da subdirectoria;
- A directoria acima da directoria corrente é identificada pelos caracteres “..”
- A directoria corrente é identificada pelo caracter “.”
- A directoria `HOME` do utilizador corrente é identificada pelo caracter “~”
- A directoria `HOME` de um utilizador é identificada por “~<nome do utilizador>”

3.3. Operações sobre ficheiros e directorias

3.3.1 Cópia de ficheiros: comando `cp`

```
cp [opções] fichOrigem1 [fichOrigem2 [...]] fichCopia
```

Notas:

- Se fichCopia já existir o seu conteúdo original será eliminado;
- Se fichCopia for o nome de uma directoria o ficheiro será copiado para dentro da directoria, mantendo o nome original
- Se forem utilizados meta-caracteres ou se for indicado mais que um ficheiro de origem, fichCopia deverá ser uma directoria. Nesse caso, todos os ficheiros manterão o nome original e serão colocados dentro da directoria indicada.

Opções mais utilizadas:

- -i : pede confirmação antes de reescrever o ficheiro de destino
- -f : força a cópia, sem pedir confirmação
- -R ou -r : copia directorias recursivamente

Para mais informações: `man cp`

3.3.2 Alteração do nome de ficheiros: comando `mv`

```
mv nomeAntigo nomeNovo
```

Notas:

- Se nomeNovo for uma directoria, o ficheiro nomeAntigo apenas muda de directoria
- O ficheiro indicado em nomeAntigo também pode ser o nome de uma directoria. Nesse caso, todo o conteúdo da directoria (incluindo subdirectorias) é movido para a nova localização, dada por nomeNovo.

3.3.3 Remoção de ficheiros: comando `rm`

```
rm [opções] ficheiros...
```

3.3.4 Visualização do conteúdo de ficheiros: comandos `cat, more e less`

```
cat ficheiros
```

Notas:

- O comando cat escreve o conteúdo dos ficheiros para o écran, por omissão. Se este conteúdo ocupar mais do que uma página do écran será necessário, regra geral bastante incómodo, utilizar as sequências ctrl^S e ctrl^Q para parar/continuar o preenchimento do écran.

- Se não for dado nenhum ficheiro como parâmetro, o `cat`, por omissão, escreve no écran o que for digitado no teclado até à digitação de **ctrl^D (indicação de fim de ficheiro).**

```
more ficheiros
less ficheiros
```

Notas:

- Estes comandos apresentam o conteúdo dos ficheiros de tal forma que o écran é preenchido página a página. A página seguinte é visualizada premindo a barra de espaços. O avanço linha a linha é feito pela tecla `<Enter>`. O caracter "q" permite terminar a visualização do ficheiro.
- Ambos os comandos dispõem de uma facilidade de pesquisa. O padrão a pesquisar é definido premindo a tecla "/" seguida do padrão, concluindo com a tecla `<Enter>`. Ocorrências seguintes do padrão são obtidas utilizando "n" em vez da barra de espaços. Os comandos permitem a utilização de padrões de pesquisa muito ricos, inclusive com recurso a meta-caracteres e a expressões regulares complexas. Informação sobre a composição destes padrões e sobre outros comandos pode ser obtida premindo h durante a utilização do programa.
- O recuo nas páginas visíveis é obtido através do comando "b".
- O comando `less` estende as opções de `more`.

3.3.5 Listar o conteúdo de directorias: comando `ls`

```
ls [opções] ficheiros...
```

Opções mais comuns:

Opção `-l`

A opção `-l` apresenta um conjunto de informação adicional para cada ficheiro listado. Por exemplo:

```
[so000@nemo tmp]$ ls –l
total 8
drwxr-xr-x  2 so000 so 48 Jul 12 17:23 aula1/
-rw-r--r--  1 so000 so 18 Jul 12 17:25 ex1.c
-rwxr--r--  1 so000 so 13 Jul 12 17:30 myls
```

A informação apresentada pelo comando `ls -l` para cada ficheiro divide-se em 7 campos:

Tipo do ficheiro

A primeira posição da sequência de caracteres inicial permite distinguir ficheiros normais de directorias. Se o primeiro caracter for (como no caso de aula1 neste exemplo) "d", o ficheiro em causa é uma directoria. Um "-" nesta posição indica tratar-se de um ficheiro regular. Um "l" apresenta um *link simbólico*, conceito que não será abordado neste documento.

Permissões

Nos 9 caracteres seguintes a existência de um "-" significa a ausência de permissão e um outro caracter a existência de permissão. As permissões em causa são:

r (read)

Permissão de leitura do ficheiro

w (write)

Permissão de escrita/eliminação do ficheiro

x (eXecute)

Permissão de execução do ficheiro

Os primeiros três caracteres descrevem as permissões para o dono do ficheiro, o segundo bloco de três caracteres as permissões para o grupo e as três últimas para os restantes utilizadores.

Estes conceitos são aprofundados na secção “alterar permissões”.

Número de *hard links*

Este conceito ultrapassa o âmbito deste documento pelo que não será explicado.

Dono do ficheiro

Utilizador que detém a propriedade do ficheiro (*owner*). É este utilizador que desfruta das permissões apresentadas no primeiro bloco. No exemplo, todos os ficheiros são propriedade do utilizador *so000*.

Grupo do ficheiro

Grupo de utilizadores atribuído ao ficheiro. São os utilizadores que pertencem a este grupo que desfrutam das permissões indicadas no segundo bloco. No exemplo, todos os ficheiros estão atribuídos ao grupo *so*.

Data/hora da última alteração

Em ficheiros alterados há mais de um ano, a hora é substituída pelo ano da última alteração.

Nome do ficheiro

Tal como apresentado na execução do comando `ls`.

Opção `-a`

A opção `-a` apresenta *todos* os ficheiros da directoria, inclusive os ficheiros iniciados por “.”. Dois ficheiros particulares nesta categoria são os ficheiros “.” e “..” que representam respectivamente a própria directoria e a directoria acima, na árvore de directorias.

3.3.6 Alterar a directoria corrente: comando `cd`

```
cd [directoria]
```

Notas:

- Se o nome da directoria for omitido, a directoria corrente passa a ser a `HOME` do utilizador.

3.3.7 Determinar a directoria corrente: comando `pwd`

```
pwd
```

3.3.8 Criar directorias: comando `mkdir`

```
mkdir directoria
```

3.3.9 Remover directorias: comando `rmdir`

```
rmdir directoria
```

Notas:

- Uma directoria só poderá ser apagada se não contiver quaisquer ficheiros ou outras subdirectorias dentro de si (comparar com `rm -r`)

3.4. Alterar permissões: comando `chmod`

3.4.1 Introdução ao conceito de permissões em unix

Em unix, as permissões podem ser definidas para o utilizador dono do ficheiro, para os outros utilizadores que pertencem ao grupo que está atribuído ao ficheiro ou para os outros utilizadores.

As permissões podem ser definidas com três modos de acesso:

- read

  Qualquer utilizador com autorização de leitura pode ver o conteúdo de um ficheiro, listando-o por exemplo através dos comandos `cat` e `more` ou utilizando um editor de texto.

  No caso de se tratar de uma directoria, o utilizador está autorizado a listar o seu conteúdo através de `ls`. Para obter informação mais detalhada acerca dos ficheiros nele contidos, tal como a que é fornecida por certas opções de `ls`, necessitará porém do direito de execução. A possibilidade de ler o conteúdo dos vários ficheiros da directoria é verificada através das permissões de cada um desses ficheiros em particular.

- write

  Um utilizador com o direito de escrita num dado ficheiro pode alterar o seu conteúdo, através, por exemplo, de um editor de texto.

  No caso de se tratar de uma directoria, o utilizador poderá alterar o seu conteúdo, apagando ou criando ficheiros nessa directoria.

- execute

  Um utilizador com direito de execução sobre um ficheiro poderá utilizá-lo a nível do interpretador de comandos como um vulgar comando de sistema, desde que as directorias acima deste possuam autorização de leitura para esse utilizador.

  No caso de se tratar de uma directoria, o utilizador pode mudar para essa directoria (comando `cd`) e copiar os seus ficheiros para outras directorias, desde que possua em relação a estas a permissão de escrita.

3.4.2 O comando `chmod`

```
chmod [opções] [ugoa...][[+-=][rwxXs-tugo...] ficheiros
chmod [opções] modo-octal ficheiros
```

A combinação das letras “ugoa” define para que utilizadores as permissões serão alteradas: (u) utilizador dono do ficheiro, (g) outros utilizadores do grupo atribuído ao ficheiro, (o) outros utilizadores, (a) todos os utilizadores.

O operador “+” adiciona a permissão, o operador “-” remove a permissão, o operador “=” faz com que o ficheiro fique apenas com as permissões que estão a ser definidas por esta invocação do comando.

As letras “rwxXstugo” indicam o modo de acesso das novas permissões: (r) leitura, (w) escrita, (x) execução. Para as outras letras consultar `man chmod`.

As permissões também podem ser definidas através de uma representação numérica na base 8 (modo-octal). A cada permissão corresponde um símbolo octal, do seguinte modo:

user/read	400

user/write	200
user/execute	100
group/read	40
group/write	20
group/execute	10
other/read	4
other/write	2
other/execute	1

Assim, por exemplo, a "`rwxr-xr-`" (todos os direitos para o dono, leitura e execução para o seu grupo e leitura para os restantes) corresponderá o número: 400+200+100+40+10+4=754

Notas:

- O dono de um ficheiro é o único utilizador que está autorizado (para além do administrador do sistema) a alterar as suas permissões.
- O grupo de um ficheiro pode ser alterado pelo seu dono através do comando `chgrp` (*change group*).
- Através do comando `ls -l` podemos visualizar as permissões associadas aos ficheiros.
- O formato modo-octal é preferível quando se pretende definir um novo conjunto de permissões para um ficheiro.
- O formato não modo-octal é tipicamente utilizado quando se pretende adicionar o remover alguma permissão às permissões já existentes.
- A opção `-R` pode ser utilizada de modo a aplicar o comando recursivamente a uma directoria.

Exemplos:

`chmod 754 fich1`

Especifica para fich1 as permissões `rwxr-xr-`" (todos os direitos para o dono, leitura e execução para o seu grupo e leitura para os restantes);

`chmod +x fich1`, `chmod a+x fich1` e `chmod ugo+x fich1`

Todos estes exemplos concedem permissão de execução ao ficheiro `fich1` a todos os utilizadores;

`chmod ug+rx,o-wx fich1 fich2`

Concede permissões de leitura e execução ao dono e ao grupo e retira permissões de escrita e execução aos outros utilizadores sobre os ficheiros fich1 e fich2;

3.5. Ocupação de espaço em disco: comando `quota`

A administração do sistema pode impor limites ao espaço ocupado por cada utilizador na sua área. Na gíria do Unix chama-se *quota* ao espaço disponível para cada utilizador. A quota de um utilizador, bem como os recursos consumidos, pode ser vista pelo comando `quota` que produz o seguinte resultado para um utilizador *bem comportado*:

```
Disk quotas for user seg002 (uid 4157):
     Filesystem blocks quota limit grace files quota limit grace
```

```
      /dev/sda8  23485 25000 35000        1545     0     0
```

E o seguinte resultado para um utilizador que já está a exceder a sua quota apesar de ainda não ter ultrapassado o limite máximo:

```
Disk quotas for user seg001 (uid 4156):
     Filesystem blocks quota limit grace files quota limit grace
      /dev/sda8 25830* 25000 35000  none  1050     0     0
```

Nota:

- Quando é utilizado um sistema de ficheiros distribuído, o comando `quota` utilizado é o da máquina local e não o da máquina remota o que dá uma noção incorrecta dos limites realmente impostos ou até a informação de que o utilizador não sofre de qualquer limitação. Para obter dados correctos será necessário que o utilizador entre na máquina (por exemplo, por `ssh`).

4. Interpretador de comandos: a *Shell*

Após a entrada numa área de trabalho, a sessão de trabalho está iniciada. Daqui em diante o sistema passará a comunicar com o utilizador através de um **programa** especial (denominado interpretador de comandos ou *shell*) cuja função é receber as directivas do utilizador (comandos) e lançar a sua execução. Terminada a execução de cada comando, a *shell* volta a indicar a sua disponibilidade apresentando o *prompt* no início de cada nova linha.

O Unix dispõe de diferentes interpretadores de comandos (*shell*s). Por exemplo `bash` (Bourne Again *Shell*), `sh`, `csh` (C-*shell*) e `tcsh` (TC-*shell*). As diferenças para os utilizadores principiantes são mínimas e praticamente irrelevantes pelo que não serão descritas neste documento. Sugere-se a consulta das páginas de manual adequadas para uma descrição mais profunda.

As áreas abertas para a disciplina de sistemas operativos estão configuradas para utilizarem a `bash`.

4.1. Estrutura da linha de comandos

O formato genérico dos comandos em Unix é:

```
comando [opcoes] {ficheiros}
```

Perante a invocação de um comando, a *shell* efectua os seguintes passos:

- Interpreta os metacaracteres – ver secção sobre metacaracteres;
- Verifica se a primeira palavra corresponde a um comando da própria *shell* - neste caso o comando é executado pela própria *shell* (ver secção *SHELL* BUILTIN COMMANDS do manual da `bash`)
- Caso contrário, o comando é pesquisado **sequencialmente na lista de directorias definidas pela variável de ambiente** `PATH`. Se o comando não for encontrado, o erro é indicado pela *shell* através da mensagem "`command not found`".

As opções são:

Letras

Por exemplo, `ls -a`.

Letras seguidas de valores

Por exemplo, `lpr -P lp1`

Cada opção é precedida do caracter "-", e é possível juntar várias opções numa cadeia. Por exemplo, para executar o comando `ls` (listagem de ficheiros) com as opções "l" e "a" poderá ser digitado `ls -la` ou `ls -l -a`.

Devido ao carácter ambíguo com que os valores associados a opções são interpretados, é conveniente que estes valores sejam sempre especificados em separado, por exemplo: `-x -G valor` em vez de `-xG valor`.

Se é cometido um erro na especificação das opções de um comando, o respectivo programa indica qual a sintaxe correcta.

Notas:

- Por omissão, o Unix não procura comandos na directoria corrente. A execução de um comando localizado na directoria corrente, quando esta não consta de `PATH` é conseguida de três formas:
    1. Qualificando o nome com a directoria corrente através do caracter "`.`" (por exemplo `./comando`);
    2. Adicionando a directoria corrente a `PATH`.
    3. Adicionando o ficheiro especial "`.`" a `PATH`. Nesse caso, qualquer que seja a directoria corrente, o sistema procurará sempre os comandos também nessa directoria.
- O Unix respeita a ordem das directorias indicadas em `PATH` na pesquisa dos comandos. De dois comandos com o mesmo nome será invocado aquele cuja localização se encontre primeiro em `PATH`. Este facto pode resultar numa quebra de segurança se de alguma forma for introduzido no `PATH` do utilizador uma directoria contendo programas alterados. Sem se aperceber, o utilizador poderá estar a executar acções de terceiros em seu nome, concedendo-lhes permissões que eles não detinham.

O caracter ";" também é um caracter terminador de comandos. Exemplo:

```
[so000@nemo so000]$ cd /
[so000@nemo /]$ ls
bin/   dev/  HOME/    lib/  opt/   root/  sys/  usr/
boot/  etc/  initrd/  mnt/  proc/  sbin/  tmp/  var/
[so000@nemo /]$ cd
[so000@nemo so000]$ cd / ; ls
bin/   dev/  HOME/    lib/  opt/   root/  sys/  usr/
boot/  etc/  initrd/  mnt/  proc/  sbin/  tmp/  var/
[so000@nemo /]$
```

O caracter "\" pode ser usado para se continuar um comando numa outra linha. Nesta linha a *shell* apresenta a *prompt* secundária. Exemplo:

```
[so000@nemo so000]$ cat /home/nemo/so/so000/trab1.entregues \
> /home/nemo/so/so000/trab2.entregues
```

4.2. Ficheiros de configuração da `bash`

Quando uma *shell* `bash` é invocada, requerendo autenticação, são executados os comandos dos seguintes ficheiros:

- `/etc/profile`
- `~/.bash_profile`, se este ficheiro não existir a `bash` executa os comandos do ficheiro `~/.bash_login`, e se este ficheiro não existir a `bash` executa os comandos do ficheiro `~/.profile`.

Quando a `bash` é invocada sem requerer autenticação são executados os comandos do ficheiro `~/.bashrc`.

Quando a `bash` termina a sua execução, invoca os comandos do ficheiro `~/.bash_logout`.

Estes ficheiros podem ser usados pelos utilizadores para personalizarem a sua *shell*.

4.3. Metacaracteres

A *shell* interpreta de forma especial vários caracteres.

Os caracteres seguintes são utilizados para especificar conjuntos de ficheiros:

- *: a *shell* procura todos os nomes de ficheiros que contenham qualquer conjunto de caracteres na posição de "*"
- ?: a *shell* procura todos os nomes de ficheiros que contenham um caracter na posição de "?"
- [ccc]: a *shell* procura todos os nomes de ficheiros que contenham um caracter pertencente a [ccc] nesta posição.

Exemplos:

```
[so000@nemo exemplo]$ ls
ex1.class  ex2.class  t1.class  t1.java~  t2.java   trabalho.java
ex1.java   ex2.java   t1.java   t2.class  t2.java~
```

//todos os ficheiros terminados com .java

```
[so000@nemo exemplo]$ ls *.java
ex1.java  ex2.java  t1.java  t2.java  trabalho.java
```

//todos os ficheiros começados por t, seguidos de um caracter e terminados em .java

```
[so000@nemo exemplo]$ ls t?.java
t1.java  t2.java
```

//todos os ficheiros começados por t e terminados por um caracter diferente de ~

```
[so000@nemo exemplo]$ ls t*[^~]
t1.class  t1.java  t2.class  t2.java  trabalho.java
```

//todos os ficheiros começados por t, seguidos de um character, um ".", um conjunto de caracteres e terminados por um caracter diferente de ~

```
[so000@nemo exemplo]$ ls t?.*[^~]
t1.class  t1.java  t2.class  t2.java
```

// todos os ficheiros começados por t, seguidos pelo caracter "1" ou pelo caracter "2", um "." e um conjunto de caracteres.

```
[so000@nemo exemplo]$ ls t[12].*
t1.class  t1.java  t1.java~  t2.class  t2.java  t2.java~
```

Os caracteres seguintes indicam à *shell* para não interpretar de forma especial determinados caracteres ou conjuntos de caracteres:

- caracter "\": indica à *shell* para não interpretar de forma especial o caracter seguinte.

```
[so000@nemo exemplo]$ echo *
ex1.class  ex1.java  ex2.class  ex2.java  t1.class  t1.java  t1.java~
t2.class t2.java t2.java~ trabalho.java
[so000@nemo exemplo]$ echo \*
*
```

- pelicas: indica à *shell* para não interpretar o conjunto de caracteres que estão entre pelicas.

```
[so000@nemo exemplo]$ echo *.*
```

```
ex1.class  ex1.java  ex2.class  ex2.java  t1.class  t1.java  t1.java~
t2.class t2.java t2.java~ trabalho.java
[so000@nemo exemplo]$ echo '*.*'
*.*
```

- aspas: indica à *shell* para não interpretar o conjunto de caracteres que estão entre aspas, excepto os caracteres $, ` e \.

```
[so000@nemo exemplo]$ echo '******a minha PATH: $PATH'
******a minha PATH: $PATH
[so000@nemo exemplo]$ echo "******a minha PATH: $PATH"
******a  minha  PATH:  /usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:usr/
lib/jdk1.4.2_06/bin
```

4.4. Sequências de controlo

O Unix, à semelhança de muitos outros sistemas utiliza um conjunto de chaves ou controlos especiais que permitem ao utilizador assinalar certas decisões e situações importantes durante o diálogo com o utilizador. As principais sequências de controlo estão listadas de seguida:

ctrl^U	Cancelamento de uma linha de comando. A totalidade da linha é eliminada, sendo de novo apresentado o *prompt* do interpretador de comandos.
ctrl^C	Terminar a execução de um programa ou de qualquer comando em execução. Pode não sortir efeito uma vez que alguns programas estão preparados para ignorar esta sequência. Caso a execução não termine tente ainda `q` seguido de `<Enter>`, ctrl^D e ctrl^Y.
ctrl^S/ ctrl^Q	Suspender o scroll do écran e voltar a activá-lo. Em alternativa deve ser usado um dos filtros `less` ou `more`.

4.5. Variáveis *shell*

As variáveis *shell* guardam informação sobre o ambiente da *shell*.

O valor de algumas variáveis é definido pela própria *shell*. Por exemplo: `HOSTNAME`.

Algumas variáveis são utilizadas pela *shell*. A *shell* atribui um valor por omissão a algumas destas variáveis. Exemplos:

- `HOME`: define a directoria `HOME` do utilizador corrente. Este valor é utilizado, por exemplo, pelo comando `cd`.
- `PATH`: conjunto de directorias onde são pesquisados comandos (a ordem é relevante). As directorias são separadas pelo caracter ":". Exemplo de um valor possível para esta variável: /usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:
- `PS1`: define o *prompt* da *shell* (por exemplo: "[\u@\h \W]\$")

  Para mais informação sobre personalização do *prompt* ver o manual da bash, secção `PROMPTING`
- `PS2`: define o *prompt* secundário.

O utilizador pode definir novas variáveis, tendo o cuidado de optar por nomes que não coincidam com variáveis usadas pela *shell*.

É possível modificar, apagar e visualizar o valor das variáveis da *shell*.

A forma de realizar as operações sobre as variáveis varia ligeiramente com a *shell* adoptada. De seguida apresentam-se os comandos da `bash` que realizam as operações sobre variáveis.

São definidos dois conceitos. VAR representa o nome da variável enquanto que $VAR representa o conteúdo da variável. Por convenção os nomes das variáveis são sempre em maiúsculas.

Operações sobre variáveis:

- A definição de uma variável é feita digitando o comando:
  `VAR=valor`
- Alterações ao seu valor são feitas da mesma forma que as definições, uma vez que não pode haver duas variáveis com o mesmo nome. Um caso particular de alteração é a concatenação ao valor já existente. Para tal, utiliza-se o conceito de `conteúdo da variável. Por exemplo, para adicionar a directoria `/users/xpto/mybin` ao fim da lista da variável `PATH` executar-se-ia:
  `PATH=$PATH:/users/xpto/mybin`
- A remoção de uma variável de ambiente é feita pelo comando `unset VAR`.
- O valor de uma variável pode ser visualizado com o comando:
  `echo $VAR`
- A lista de variáveis definidas pode ser visualizada pelos comandos `set` ou `env`.
- Quando uma variável de ambiente é definida, o seu âmbito restringe-se à *shell* corrente e não a *shell*s ou processos filhos que venham a ser criados no seu âmbito. O comando `export VAR` propaga a definição da variável a todos os processos filhos que venham a ser criados.

Exemplo:

```
[so000@nemo exemplo]$ X=ola
[so000@nemo exemplo]$ echo $X
ola
[so000@nemo exemplo]$ X=hello
[so000@nemo exemplo]$ echo $X
hello
[so000@nemo exemplo]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:
[so000@nemo exemplo]$ PATH=$PATH:/HOME/nemo/seg/so000/bin
[so000@nemo exemplo]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/HOME/nemo/seg/so000/bin
[so000@nemo exemplo]$ PATH=$PATH:
[so000@nemo exemplo]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/HOME/nemo/seg/so000/bin:
[so000@nemo exemplo]$ unset X
[so000@nemo exemplo]$ echo $X

[so000@nemo exemplo]$ X=hello
[so000@nemo exemplo]$ bash
[so000@nemo exemplo]$ echo $X

[so000@nemo exemplo]$ exit
[so000@nemo exemplo]$ export X
[so000@nemo exemplo]$ bash
[so000@nemo exemplo]$ echo $X
hello
```

Notas:

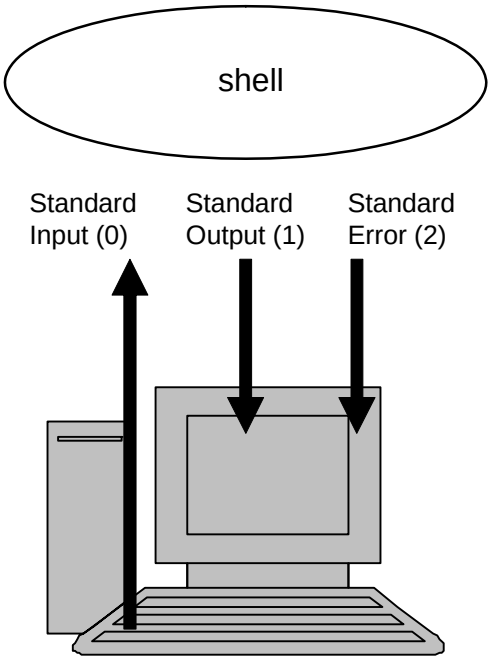
- Quando adicionamos a directoria corrente à variável `PATH` o caracter "." pode ser omitido. Exemplos:

```
PATH=:/bin:/usr/bin    # idêntico a PATH=.:/bin:/usr/bin
PATH=/bin::/usr        # idêntico a PATH=/bin:.:/usr/bin
PATH=/bin:/usr:        # idêntico a PATH=/bin:/usr/bin:.
```

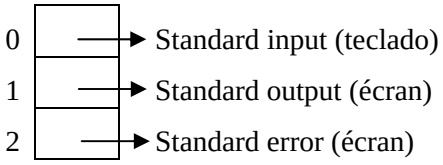
- As variáveis definidas nos ficheiros de configuração da *shell* devem ser exportadas.

4.6. Redirecção de entradas e saídas

A *shell* assume por omissão que os seus ficheiros de entrada e saída são respectivamente o teclado e o écran.



Para cada processo em execução existe uma tabela de ficheiros abertos. Nesta tabela são sempre inseridos automaticamente três descritores:



Estes descritores podem ser redireccionados:

- < redirecciona a entrada
- > redirecciona a saída
- >> redirecciona a saída para acrescento
- 2> redirecciona a saída para mensagens de erro
- 2> redirecciona a saída para mensagens de erro para acrescento
- n> redirecciona o descrito n

Exemplos:

```
[so000@nemo so000]$ echo conteudo da root: >ls.root
[so000@nemo so000]$ ls / >>ls.root
[so000@nemo so000]$ cat ls.root
conteudo da root:
bin/
boot/
dev/
etc/
HOME/
initrd/
```

```
lib/
mnt/
opt/
proc/
root/
sbin/
sys/
tmp/
usr/
var/
```

```
[so000@nemo so000]$cat >nTrabalhos
trab1: 50
trab2: 48
trab3: 48
<ctrl^d>
[so000@nemo so000]$cat nTrabalhos
trab1: 50
trab2: 48
trab3: 48
```

```
[so000@nemo exemplo]$ echo exemplo de redireccionamento > file1
[so000@nemo exemplo]$ ls
file1
[so000@nemo exemplo]$ cat file1 file2
exemplo de redireccionamento
cat: file2: No such file or directory
[so000@nemo exemplo]$ cat file1 file2 2>erros
exemplo de redireccionamento
[so000@nemo exemplo]$ cat erros
cat: file2: No such file or directory
[so000@nemo exemplo]$
```

É possivel associarem-se redireccionamentos:

```
[so000@nemo exemplo]$ cat file1 file2 >outfile 2>&1
[so000@nemo exemplo]$ cat outfile
exemplo de redireccionamento de erros
cat: file2: No such file or directory

[so000@nemo exemplo]$ cat file1 file2 >outfile 2>outfile
[so000@nemo exemplo]$ cat outfile
cat: file2: No such file or directory

[so000@nemo exemplo]$ cat file2 file1 >outfile 2>outfile
[so000@nemo exemplo]$ cat outfile
exemplo de redireccionamento de erros
```

A sequência de caracteres "<<" é utilizada na construção de *here documents*. O formato é o seguinte:

```
<<palavra
```

as linhas existentes entre "`<<palavra`" e a próxima ocorrência de `palavra` são utilizadas como entradas (stdin) para um comando.

```
[so000@nemo exemplo]$ cat <<ola
> exemplo
> de
> here document
> ola
exemplo
```

```
de
here document
[so000@nemo exemplo]$
```

4.6.1 Utilização de pipes: caracter "|"

Um *pipeline* é uma sequência de comandos separados pelo caracter "|". Deste modo o *standard output* (resultados) do primeiro comando é redireccionado para o *standard input* (entradas) do segundo comando, e assim sucessivamente.

Exemplo:

```
[so000@nemo exemplo]$ ls
erros  file1  outfile
[so000@nemo exemplo]$ ls |wc -w
3
```

Notas:

- O resultado do comando `ls` é redireccionado para o *standard input* do comando `wc`.

4.7. Facilidades da *shell*

4.7.1 Completar texto na linha de comandos – caracter TAB

A `bash` tenta completar o texto da linha d comando da seguinte forma:

- Se o texto começa com $, a `bash` tenta completar com o nome de uma variável;
- Se o texto começa com ~, a `bash` tenta completar com o nome de um utilizador;
- Senão tenta completar com um comando
- Exemplo

```
[so000@nemo so000]$ h <tab>
a shell completa com o comando history:
[so000@nemo so000]$ history
```

- Senão tenta completar com um nome de um ficheiro/directoria.
- Exemplo

```
[so000@nemo so000]$ De <tab>
a shell completa com a directoria Desktop:
[so000@nemo so000]$ Desktop/
```

Informação adicional: man complete e man `bash`

4.7.2 Comando `history`

O comando history permite obter a lista dos comandos previamente invocados.

Alguns exemplos de utilização da informação disponibilizada pelo histórico:

- !! : refere o último comando invocado
- !n : refere o n-ésimo comando do histórico
- !<texto> : refere o último comando cujo início é <texto>
- Podem também ser utilizadas as setas para percorrer o histórico dos comandos

5. Processos

Cada utilizador de um sistema Unix tem associados vários processos. Define-se como processo um programa em execução. Sempre que a nível da *shell* se invoca um comando, são criados um

ou mais processos no sistema, para a sua execução. Cada processo tem um identificador único, **o pid do processo.**

5.1. Visualizar processos: comando `ps`

A execução do comando `ps` sem opções adicionais apresenta informação sobre os processos associados ao utilizador e ao terminal correntes. A informação apresentada inclui: o pid do processo, o terminal associado ao processo, o tempo de CPU utilizado e o nome do comando que esteve na origem do processo. Exemplo:

```
  PID TTY          TIME CMD
27255 pts/3    00:00:00 bash
30690 pts/3    00:00:00 ps
```

Algumas opções do comando `ps`:

- -e: apresenta informação sobre todos os processos
- -f, -F: apresentam mais informação sobre os processos (ver `man ps`)
- -H: apresenta a hierarquia dos processos
- -L: apresenta informação sobre *threads*

5.2. Execução de comandos em *background*

Normalmente a execução de comandos é sequencial: a *shell* cria um processo filho para executar o comando, e aguarda que o processo termine para poder prosseguir.

No entanto, a *shell* permite que os comandos sejam lançados em *background*, através do operador "&". Deste modo, a *shell* fica disponível para tratar outros comandos.

Exemplo:

```
[so000@nemo so000]$ emacs trab1.c &
[1] 30823
[so000@nemo so000]$ ps
  PID TTY          TIME CMD
27255 pts/3    00:00:00 bash
30823 pts/3    00:00:00 emacs
30824 pts/3    00:00:00 ps
```

Notas:

- O lançamento de comandos em *background* liberta imediatamente a *shell* para poder continuar a tratar comandos;
- A *shell* apresenta o pid do processo;
- Quando o processo terminar, a *shell* avisará o utilizador através da seguinte mensagem: `[id]+ Done comando`
- Quando um processo é executado em *background*, normalmente tem-se stdin = /dev/null e stdout = /dev/tty

5.3. Matar um processo: comando `kill`

É frequente acontecer que a execução de um programa dê origem a uma situação da qual parece não haver saída pelos meios mais pacíficos.

O comando `kill` envia um determinado sinal para um processo ou grupo de processos.

Exemplos de utilização:

```
[so000@nemo so000]$ kill 30823
```

```
[so000@nemo so000]$ kill -9 30823
[1]+  Killed                  emacs trab1.c
[so000@nemo so000]$ ps
  PID TTY          TIME CMD
27255 pts/3    00:00:00 bash
30975 pts/3    00:00:00 ps
```

Notas:

- No primeiro caso o comando `kill` envia o sinal `TERM` (número 15) ao processo. Por omissão, quando o processo recebe este sinal morre. No entanto, o processo pode tratar este sinal, por exemplo, ignorando-o.
- No segundo caso o comando `kill` envia o sinal `KILL` (número 9) ao processo. Este sinal não pode ser tratado pelos processos. Quando um processo recebe este sinal morre.
- A sequência de caracteres ctrl^c envia o sinal `SIGINT` (número 2) ao processo. Por omissão, quando os processos recebem este sinal, morrem. No entanto, este sinal também pode ser tratado pelos processos.
- Um utilizador só pode matar processos que tenham sido lançados por si próprio. O administrador do sistema pode realizar esta operação sobre qualquer processo em execução.

Mais informações sobre o comando `kill`: `man kill`.

Mais informações sobre sinais: `man -s 7 signal`.

5.4. Controlo de *Jobs*

A *shell* associa um *job* a cada comando ou a cada conjunto de comandos separados pelo caracter "|" e mantém uma lista dos *jobs* que estão em execução.

O controlo de *jobs* permite parar a execução de processos e continuar a sua execução posteriormente. Com este objectivo, a `bash` disponibiliza as seguintes funcionalidades:

- O comando `jobs` permite visualizar a lista dos *jobs* em execução.
- A sequência de caracteres ctrl^z permite parar um processo e colocá-lo em *background.*
- A sequência de caracteres ctrl^y permite parar o processo quando este tentar ler dados a partir do terminal.
- O comando `bg <número do job>` permite continuar a execução do processo em *background*.
- O comando `fg <número do job>` permite continuar a execução do processo em *foreground* (`fg 1` é idêntico a **`%1`**).

Exemplo:

```
[so000@nemo exemplo]$ vi trab1.c
// dentro do vi faço ctrl^z
[1]+  Stopped                 vi trab1.c
[so000@nemo exemplo]$ sleep 120 &
[2] 32385
[so000@nemo exemplo]$ jobs
[1]+  Stopped                 vi trab1.c
[2]-  Running                 sleep 120 &
[so000@nemo exemplo]$ sleep 130
ctrl^z
[3]+  Stopped                 sleep 130
[so000@nemo exemplo]$ jobs
[1]-  Stopped                 vi trab1.c
```

```
[2]   Running                 sleep 120 &
[3]+  Stopped                 sleep 130
[so000@nemo exemplo]$ bg 3
[3]+ sleep 130 &
[so000@nemo exemplo]$ jobs
[1]+  Stopped                 vi trab1.c
[2]   Running                 sleep 120 &
[3]-  Running                 sleep 130 &
[so000@nemo exemplo]$ %1
//volto a poder utilizar o vi
```

6. Outros comandos

6.1. Manual do unix: comando `man`

O comando man permite visualizar as páginas do manual do Unix.

Modo de utilização do comando:

```
man [opções] comando
```

Se houver informação respeitante a `comando`, `man` apresentará no écran o texto dessa informação dividido em três secções (tal como no manual de Unix).

A secção *Name* descreve sumariamente o comando. Na secção *Synopsis* é apresentada a sintaxe de invocação. Finalmente a secção *Description* explica com maior detalhe o comando, descrevendo os seus argumentos, opções e ficheiros. Uma secção opcional de grande interesse é *See also* que encaminha o utilizador para as páginas de manual de funções relacionadas.

Para facilitar a leitura, toda a informação é apresentada página a página. O avanço nas páginas é feito pela barra de espaços. A terminação de visualização, pesquisa e opções de recuo na página estão dependentes do filtro de controlo de visualização utilizado: `more`, `less` ou nenhum (**ver notas sobre estes comandos**).

As páginas do manual estão organizadas em secções:

- Secção 1: comandos
- Secção 2: chamadas ao sistema operativo (funções escritas na linguagem de programação C)
- Secção 3: funções de biblioteca
- Secção 4: ficheiros especiais – device files
- Secção 5: formatos comuns de ficheiros
- Secção 6: jogos
- Secção 7: miscelanea
- Secção 8: comandos de administração
- Secção 9: funções da interface do núcleo do unix

Alguns comandos (por exemplo, *printf*) dispõem de mais de uma página de manual, dispersas por várias secções. Por omissão, `man` executa a pesquisa apenas até localizar a primeira ocorrência para o comando. Este comportamento pode ser alterado recorrendo a duas opções:

- A **opção `-a`** pesquisa em todas as secções a página referente ao comando. Se existir mais que uma página, a seguinte será apresentada quando a visualização da anterior for terminada.

- A **opção `-S`** aceita como argumento um conjunto de secções separadas pelo caracter dois pontos (:). Pode-se por isso fazer `man -S 3 printf` para visualizar a página referente ao comando `printf` da secção 3 em vez da referente ao comando na secção 1 que seria apresentada por omissão.

A utilização normal do comando `man` pressupõe que o utilizador conhece o nome do comando sobre o qual quer obter informação. Caso contrário, o utilizador pode utilizar a seguinte opção:

- **Opção `-k`** (idêntica ao comando `apropos`) procura a palavra fornecida como argumento numa base de dados de comandos e suas descrições. Desta forma, o comando `man` apresenta uma lista de comandos e suas descrições (semelhante à secção *Name* do comando `man`) em que a palavra indicada figura no nome do comando ou na sua descrição.

6.2. Filtros

Os filtros são comandos que lêem alguns dados de entrada, realizam algumas operações sobre esses dados e escrevem os respectivos resultados.

Exemplos de filtros:

- `fgrep, grep, egrep`: procuram as linhas que contêm um determinado padrão
- `wc`: conta o número de linhas, palavras e letras
- `sort`: ordena linhas de texto
- `tail`: apresenta o última parte do ficheiro
- `uniq`: remove linhas duplicadas de um ficheiro ordenado
- `cut`: remove secções de cada linha dos ficheiros
- `sed, awk`: filtros programáveis

6.2.1 Filtros `grep, egrep, fgrep`

`grep [opção] padrão [ficheiro...]`

normalmente o padrão tem de ser uma expressão regular “básica”

`egrep [opção] padrão [ficheiro...]`

o padrão poder ser uma expressão regular estendida

`fgrep [opção] palavra [ficheiro...]`

considera apenas o valor literal de palavra

Algumas expressões regulares usadas com `grep`:

- \c: literalmente o valor de c
- ^: no início da linha
- $: no fim da linha
- .: qualquer caracter
- [...]: qualquer caracter contido em ...
- [^...]: qualquer caracter não contido em ...
- r*: zero ou mais ocorrências de r

Algumas extensões de `egrep`:

- r+: uma ou mais ocorrências de r
- r?: zero ou uma ocorrência de r
- r1|r2: r1 ou r2

Exemplos:

```
[so000@nemo exemplo]$ cat exemplo.grep
Ficheiro para exemplicar a utilização dos greps
linha 1
linha 2
^linha           3
esta é a linha quatro
e esta é a linha cinco

[so000@nemo exemplo]$ grep linha exemplo.grep
linha 1
linha 2
^linha           3
esta é a linha quatro
e esta é a linha cinco
[so000@nemo exemplo]$ grep ^linha exemplo.grep
linha 1
linha 2
```

//no exemplo seguinte o caracter \ é interpretado pela shell

```
[so000@nemo exemplo]$ grep \^linha exemplo.grep
linha 1
linha 2
[so000@nemo exemplo]$ grep '\^linha' exemplo.grep
^linha           3
[so000@nemo exemplo]$ grep s exemplo.grep
Ficheiro para exemplicar a utilização dos greps
esta é a linha quatro
e esta é a linha cinco
[so000@nemo exemplo]$ grep s$ exemplo.grep
Ficheiro para exemplicar a utilização dos greps
[so000@nemo exemplo]$ grep 'linha [0-9]' exemplo.grep
linha 1
linha 2
[so000@nemo exemplo]$ grep 'linha +[0-9]' exemplo.grep
[so000@nemo exemplo]$ egrep 'linha +[0-9]' exemplo.grep
linha 1
linha 2
^linha           3
[so000@nemo exemplo]$ grep 'linha .' exemplo.grep
linha 1
linha 2
^linha           3
esta é a linha quatro
e esta é a linha cinco
[so000@nemo exemplo]$ egrep 'linha .+o$' exemplo.grep
esta é a linha quatro
e esta é a linha cinco
[so000@nemo exemplo]$ egrep 'linha .o$' exemplo.grep
[so000@nemo exemplo]$ egrep 'linha .+o$' exemplo.grep
esta é a linha quatro
e esta é a linha cinco
[so000@nemo exemplo]$ egrep 'linha.+o$' exemplo.grep
esta é a linha quatro
e esta é a linha cinco
```