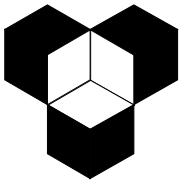


Escola Superior de
Tecnologia e de Gestão

INSTITUTO POLITÉCNICO DE BRAGANÇA

Introdução aos Sistemas Informáticos

2004/2005

Engenharia Informática — Engenharia Mecânica —

Engenharia Química — Gestão e Engenharia Industrial

O Sistema Operativo Linux

Autor: Reis Lima Quarteu (reis.quarteu@ipb.pt)

Outubro de 2004

Índice

1. Comandos Básicos

1.1 Para Sair do Sistema

Uma vez que esteja conectado ao sistema operativo *Linux* (isto é, que tenha iniciado a sua sessão de trabalho), há-de chegar o momento em que terá, ou em que quererá, desligar-se do sistema operativo (ou seja, terminar a sua sessão de trabalho). Para isso, existem 3 possibilidades diferentes:

- O comando `exit`;
- O comando `logout`;
- A combinação de teclas `Ctrl+D`, ou `^D`.

Deve-se ter em atenção que o sistema operativo *Linux* diferencia as letras maiúsculas e as letras minúsculas. Assim, se escrever `"exit"`, será executado o comando que permite ao utilizador desligar-se do sistema operativo; mas se escrever coisas como `"Exit"`, `"EXIT"`, `"eXit"`, etc., o resultado será um erro: o sistema operativo *Linux* informará o utilizador de que o comando indicado não existe. Esta restrição também se verifica nos nomes dos ficheiros e das directorias.

1.2 Gestão da Palavra-Passe

Outro comando básico é o que permite alterar a sua palavra-passe: `passwd`. Ao executar este comando, é-lhe pedido a sua actual palavra-passe. Isto é muito importante nas situações em que, estando a sua sessão de trabalho aberta, você tiver que se ausentar momentaneamente do computador em que estiver a trabalhar. Nada lhe garante que o "vizinho do lado" não lhe tente "pregar uma partida", alterando-lhe a sua palavra-passe durante a sua ausência. Ao pedir a sua actual palavra-passe antes de permitir que uma nova seja introduzida, o sistema garante a sua segurança em situações como a anterior (que não são tão raras como isso...). Depois de introduzir a sua palavra-passe actual, é necessário escrever a nova palavra-passe duas vezes. O sistema só aceita a nova palavra-passe se as duas cópias forem exactamente iguais.

1.3 Informação sobre os Utilizadores

O sistema operativo *Linux* oferece-lhe a possibilidade de obter alguma informação sobre os utilizadores que estejam a trabalhar no sistema em simultâneo consigo. Para isso, dispõe de três comandos: `who`, `finger` e `w`.

Para cada utilizador ligado ao sistema, o comando `who` mostra-nos o seu nome de utilizador (*"login"*), o tipo de máquina através da qual o utilizador está ligado ao sistema, a hora em que o utilizador se ligou ao sistema, e o nome da máquina através da qual o utilizador está ligado ao sistema. Um exemplo da execução do comando `who` pode ser examinado a seguir:

```
root             pts/0     Dec 27 11:30     janus.ccom.ipb.pt
reis.quarteu     pts/1     Dec 30 11:40     fwdoc.estig.ipb.pt
nuno             pts/3     Dec 29 17:54     charon.ccom.ipb.pt
```

Na 2.ª coluna da lista (tipo de máquina), a sigla `pts` significa *"Personal Terminal System"* (Sistema Terminal Pessoal), a qual representa um PC. O número que aparece à direita de `"pts"` é atribuído pelo próprio sistema operativo à medida que utilizadores diferentes se ligam ao sistema através de diferentes computadores pessoais.

Se acrescentarmos ao comando `who` os argumentos especiais `"am i"`, o sistema disponibilizará informação sobre o utilizador da sessão actual de trabalho, isto é, você! Essa informação consiste no nome completo da máquina à qual você se encontra ligado ao sistema, seguida de um ponto de exclamação ("!"), seguida do seu nome de utilizador (*"login"*); o tipo de máquina através da qual você está ligado ao sistema, a hora em que você se ligou ao sistema, e o nome da máquina através da qual você está ligado ao sistema. Um exemplo da execução do comando `who am i` pode ser examinado a seguir:

```
elara.ipb.pt!reis.quarteu   pts/1   Dec 30 11:40   fwdoc.estig.ipb.pt
```

Outro comando que permite obter informações sobre os utilizadores ligados ao sistema é o `finger`. Essas informações são as seguintes:

- Nome de utilizador (*"login"*);
- Nome próprio do utilizador (o seu nome, tal como está registado no sistema);

- Tipo de máquina através da qual o utilizador está ligado ao sistema;
- Tempo de inactividade na sessão actual de trabalho;
- Hora em que o utilizador se ligou ao sistema;
- Nome da máquina através da qual o utilizador está ligado ao sistema[1].

Um exemplo da execução do comando `finger` pode ser examinado a seguir:

```
Login          Name             Tty     Idle    Login Time
nuno           Nuno Goncalves   pts/3   19:39   Dec 29 17:54   charon.ccom.ipb.pt
reis.quarteu   Reis Quarteu     pts/1           Dec 30 14:37   fwdoc.estig.ipb.pt
root           Root             pts/0   2d      Dec 27 11:30   janus.ccom.ipb.pt
```

Por fim, o comando `w` também fornece informação sobre os utilizadores actualmente ligados ao sistema, além de dizer o que esses utilizadores estão a fazer (isto é, qual o comando que cada utilizador está a executar no momento da execução do comando `w` na nossa máquina). Essas informações são as seguintes:

- Nome de utilizador (*"login"*);
- Tipo de máquina através da qual o utilizador está ligado ao sistema;
- Nome da máquina através da qual o utilizador está ligado ao sistema;
- Hora em que o utilizador se ligou ao sistema;
- Tempo de inactividade na sessão actual de trabalho;
- JCPU e PCPU[2];
- Comando actualmente em execução.

Um exemplo da execução do comando `w` pode ser examinado a seguir:

[1] Se disponível, também poderá aparecer o nome e o telefone do gabinete do utilizador. Essa informação pode ser interessante se o sistema operativo *Linux* for utilizado pela rede de computadores de uma grande organização.
[2] JCPU representa o tempo total de execução consumido por todos os processos executados a partir da máquina através da qual o utilizador se encontra ligado ao sistema; PCPU representa o tempo total de execução consumido pelo processo que cada utilizador tem actualmente em execução.

```
USER          TTY     FROM                LOGIN@   IDLE     JCPU    PCPU    WHAT
root          pts/0   janus.ccom.ipb.pt   Fri11am  2 days   0.03s   0.03s   -bash
reis.quarteu  pts/1   fwdoc.estig.ipb.pt  2:37pm   0.00s    0.05s   0.02s   w
nuno          pts/3   charon.ccom.ipb.pt  Sun 5pm  4:15     0.54s   0.06s   -bash
```

1.4 Data do Sistema

Para saber a data actual do sistema, deve utilizar o comando `date`. Ao contrário de outros sistemas operativos, este comando do *Linux* apenas permite que se consulte a data, mas não permite a sua alteração. Se isso fosse possível, todos os utilizadores poderiam alterar a data do sistema à sua vontade, o que traria consequências imprevisíveis sobre o funcionamento do sistema operativo e dos programas em execução. O único utilizador que poderá alterar a data do sistema é o super-utilizador.

O super-utilizador (também conhecido por supervisor, administrador ou pela denominação inglesa *"root"*) é um utilizador especial que possui privilégios especiais de acesso aos recursos do sistema operativo. É também o responsável pela administração, configuração e manutenção do sistema operativo. É ele quem permite que um dado utilizador possa utilizar os recursos disponibilizados pelo sistema operativo. Em suma, o super-utilizador é o máximo responsável de uma instalação do sistema operativo *Linux* numa determinada máquina. Em consequência, apenas ao super-utilizador é permitido a actualização da data do sistema.

1.5 Manuais

O sistema operativo *Linux* disponibiliza a todos os utilizadores ampla documentação sobre todos os seus comandos. Essa documentação está disponível através do comando `man`. Se quisermos consultar a informação disponível sobre um determinado comando do sistema operativo, basta-nos escrever "`man`", seguido do comando que queremos consultar. Por exemplo, se quisermos consultar o "manual" do comando `date`, apenas teremos que introduzir o comando seguinte:

```
man date
```

A informação dos manuais do *Linux* é um tanto ou quanto técnica para quem apenas pretenda aprender a maneira correcta de trabalhar com um determinado comando. Mas, com

um pouco de paciência e persistência, encontrará nesses manuais toda a informação necessária para tirar o máximo partido das funcionalidades disponibilizadas pelo sistema operativo!

Para que se possa visualizar o manual de um comando da maneira mais cómoda possível, é necessário utilizar algumas teclas:

- Uma linha para a frente: *"enter"* ou "J";
- Uma linha para trás: "K";
- Uma página para a frente: barra de espaços;
- Meia página para a frente: "D";
- Meia página para trás: "U";
- Sair do manual: "Q".

2. Gestão de Ficheiros e Directorias

2.1 A Árvore de Directorias

A directoria principal da árvore de directorias do sistema operativo *Linux* chama-se directoria raiz e é representada pelo caracter "/"[3]. Dentro dessa directoria, existem habitualmente as seguintes subdirectorias:

`bin`	Contém um conjunto mínimo de programas utilitários, que são usados durante o arranque do sistema.
`boot`	Arranque do sistema: contém um ficheiro com o núcleo do sistema operativo e vários ficheiros auxiliares.
`dev`	Contém ficheiros que representam todos os dispositivos de *hardware* e periféricos do sistema (*"device drivers"*).
`etc`	Contém a maioria dos ficheiros de configuração do sistema operativo.
`home`	Directorias de trabalho (ou directorias pessoais) dos utilizadores.
`tmp`	Ficheiros temporários: todos os utilizadores podem criar ficheiros de dados temporários nesta directoria. Estes ficheiros podem ser apagados sempre que o sistema arranca.
`usr`	Contém mais subdirectorias com programas, bibliotecas, utilitários, documentação, etc.

Cada utilizador possui uma directoria pessoal dentro do sistema operativo. Portanto, e de acordo com a tabela anterior, essa directoria está localizada dentro da directoria `/home`. Na máquina utilizada durante as aulas dedicadas ao estudo do sistema operativo *Linux*, uma das subdirectorias da directoria `/home` tem o nome de `estig`[4], dentro da qual se encontram novas subdirectorias, algumas das quais representam todos os cursos ministrados pela ESTiG/IPB. No que diz respeito à disciplina de **Introdução aos Sistemas Informáticos**, interessa-nos referir as subdirectorias que se seguem: `ei` (Engenharia Informática), `em` (Engenharia Mecânica), `eq` (Engenharia Química) e `gei` (Gestão e Engenharia Industrial). Finalmente, cada uma dessas directorias contém novas subdirectorias: uma para cada aluno

[3] É necessário ter em atenção que este caracter é diferente do caracter utilizado pelos sistemas operativos MS-DOS e *Windows*, da Microsoft. Enquanto que o *Linux* representa a directoria raiz por "/", os sistemas da Microsoft representam-na por "\".
[4] ESTiG é a sigla da Escola Superior de Tecnologia e de Gestão do Instituto Politécnico de Bragança.

matriculado nesses cursos. Os nomes dessas subdirectorias são os números de identificação de cada aluno dentro do IPB. Estas são as directorias pessoais de cada um dos alunos daqueles cursos.

A árvore de directorias da máquina descrita no parágrafo anterior pode ser representada pela figura **1**:

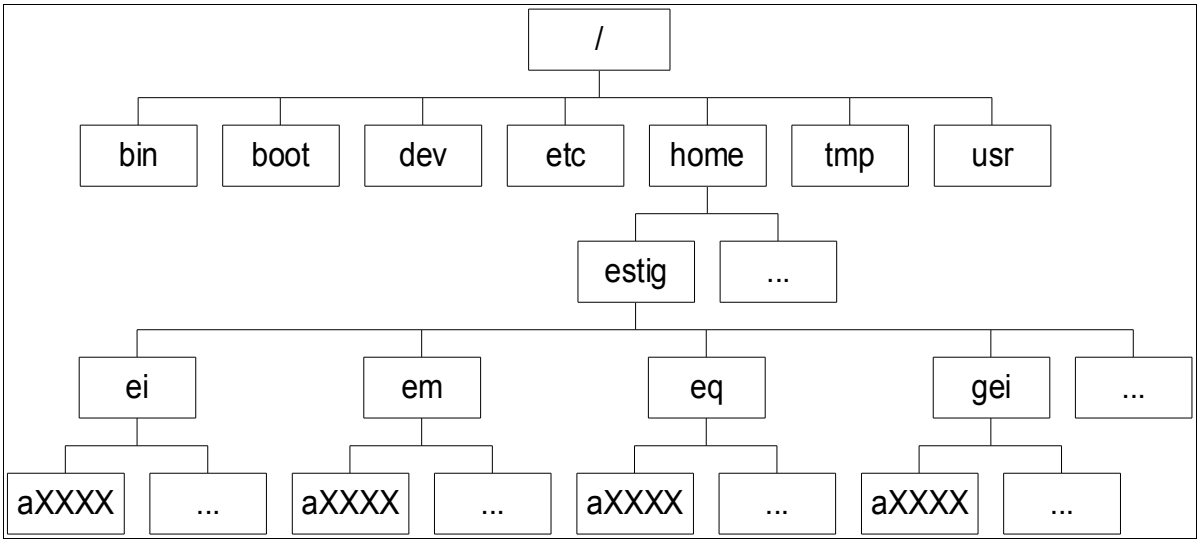


Figura 1: *Árvore de directorias de uma máquina Linux. Dentro da directoria /home/estig encontram-se subdirectorias referentes a cada um dos cursos da ESTiG; dentro dessas directorias encontram-se as directorias pessoais dos alunos desses cursos.*

2.2 Caminhos

Quando um utilizador inicia uma nova sessão de trabalho, o sistema operativo *Linux* posiciona-o na sua directoria pessoal. Esta é a primeira directoria corrente (ou actual) de um utilizador. Todos os comandos que o utilizador executar sê-lo-ão no contexto da directoria corrente do utilizador, a menos que ele explicitamente indique outra directoria. Para isso, é necessário saber como indicar os caminhos que levam de uma directoria a uma outra qualquer. À directoria que em cada momento se pretende referenciar através de um caminho chamar-se-á **directoria destino**.

Em essência, existem dois tipos de caminhos: **absolutos** e **relativos**. Os caminhos absolutos são formados pelos nomes de todas as directorias entre a directoria raiz e a directoria destino, inclusive, separados pelo caracter "/". Por outras palavras, os caminhos absolutos começam sempre pela directoria raiz. Por exemplo: de acordo com a árvore de

directorias representada na figura **1**, o caminho absoluto para a directoria pessoal do utilizador **eq1234** é o seguinte[5]:

```
/home/estig/eq/eq1234
```

Num caminho absoluto, o primeiro caracter "/" representa a directoria raiz. Todos os caracteres "/" restantes servem apenas para separar os nomes das várias directorias que constituem o caminho.

Por outro lado, os caminhos relativos são formados pelos nomes de todas as directorias entre a directoria actual e a directoria destino, inclusive. Também aqui os nomes das directorias são separados pelo caracter "/". Por exemplo, se a directoria actual for `/home/estig`, então o caminho relativo entre essa directoria e a directoria pessoal do utilizador **eq1234** do exemplo anterior é o seguinte:

```
eq/eq1234
```

Este é um exemplo de um caminho relativo que permite ir de uma directoria para uma directoria destino que se encontra dentro da própria directoria inicial. Em situações como esta, apenas é necessário indicar os nomes das diversas directorias que constituem o caminho, uma vez que cada directoria do caminho é uma subdirectoria da directoria anterior[6].

Porém, há muitas situações em que acontece exactamente o contrário: a directoria actual encontra-se "dentro" da directoria destino. Nesse caso, é necessário referenciar a directoria-mãe[7] da directoria actual. Mas tal não é possível colocando o nome da directoria-mãe no caminho relativo que se quer escrever, porque a directoria-mãe não é uma subdirectoria da directoria actual. Assim, torna-se necessário utilizar uma notação especial para representar uma directoria-mãe. Essa notação é representada por dois pontos: "**..**". Por exemplo, se a directoria actual for `/home/estig/eq`, então o caminho relativo para a directoria `/home/estig` é representada simplesmente por:

```
..
```

[5] Para uma melhor compreensão, os exemplos que se seguem (sobre caminhos numa árvore de directorias) serão baseados no esquema representado na figura **1**.
[6] A primeira directoria que aparece num caminho relativo é uma subdirectoria da directoria actual.
[7] A directoria-mãe é aquela que contém uma directoria. Por exemplo, na árvore de directorias representada na figura **1**, a directoria `estig` é a directoria-mãe da directoria `eq`.

Como é natural, é possível construir caminhos que levem a directorias que estejam ainda mais acima na árvore de directorias do sistema operativo[8]. Por exemplo, se a directoria actual for, tal como no exemplo anterior, `/home/estig/eq`, então o caminho relativo para a directoria `/home` é:

```
../..
```

Este caminho relativo mostra que é necessário subir dois níveis na hierarquia da árvore de directorias para se chegar à directoria destino.

Note-se que a única directoria da árvore de directorias do sistema operativo que não possui uma directoria-mãe é a directoria raiz.

É possível combinar "subidas" e "descidas" na árvore de directorias para se ir de uma directoria a outra. Por exemplo, se a directoria actual continuar a ser `/home/estig/eq`, então o caminho relativo para a directoria `/home/estig/ei` é:

```
../ei
```

Existem situações em que se torna necessário referenciar explicitamente a directoria actual no caminho relativo que se pretende escrever. Nestas situações, não se pode simplesmente escrever o nome da directoria actual, porque ela não é uma subdirectoria de si própria. Também não se pode usar a notação "**..**", porque a directoria actual não é a directoria-mãe de si própria! Assim, torna-se necessário utilizar uma outra notação especial para representar a directoria actual. Essa notação é representada por um único ponto: "**.**".

Com estas notações, é possível escrever caminhos muito complicados, o que torna esta notação de representação de caminhos entre directorias bastante poderosa. Considerem-se os três exemplos que se seguem.

Se a directoria actual for `/home/estig/eq/eq1234`, um caminho relativo que representa a própria directoria actual (**.**), que sobe até à sua directoria-mãe (**..**) e que volta à directoria `eq1234` é perfeitamente válido, apesar de ter pouco interesse. Tal caminho relativo conduz-nos da directoria actual, descrita acima, até à mesma directoria!

```
./../eq1234
```

[8] Diz-se que uma directoria-mãe se encontra "acima" na árvore de directorias do sistema operativo.

Outro exemplo, agora com um caminho absoluto. Aqui, definir-se-á um caminho que partindo da directoria raiz, entre na directoria `temp`, volte para trás, entre na directoria `home`, até chegar à directoria `eq1234`. Este exemplo mostra que é perfeitamente possível referenciar a directoria-mãe de uma qualquer directoria dentro de um caminho absoluto.

```
/temp/../home/estig/eq/eq1234
```

Um último exemplo consiste em, sendo `/home/estig` a directoria actual, subir duas vezes na árvore das directorias, descer até à directoria `/home/estig/eq/eq1234`, subir até à directoria-mãe desta última directoria, e voltar a descer para a directoria `eq1234`. Mais uma vez, trata-se de um exemplo sem grande importância prática, mas que permite verificar as possibilidades disponíveis para a construção de caminhos entre directorias.

```
../../home/estig/eq/eq1234/../eq1234
```

Muitas vezes, há a necessidade de escrever um caminho que conduza desde a directoria actual, qualquer que esta seja, até à directoria pessoal do utilizador. O sistema operativo *Linux* disponibiliza uma notação para representar a directoria pessoal de cada utilizador: trata-se do acento gráfico "~" (til). Esta notação apenas poderá ser utilizada em caminhos relativos, que passam assim a principiar na directoria pessoal do utilizador. Por exemplo, se a directoria actual for `/temp`, o utilizador pode definir o caminho para uma directoria `docs`, que se encontra dentro da sua directoria pessoal, da seguinte maneira:

```
~/docs
```

Qualquer que seja a directoria destino a que se queira chegar, existem vários caminhos que podem ser utilizados: a partir da directoria raiz, da directoria actual ou da directoria pessoal. O caminho que deverá ser utilizado depende de cada situação: uma boa regra que se deve seguir é utilizar o caminho que nos obrigue a escrever menos! Não nos esqueçamos que estamos a indicar, através do teclado, o que queremos que seja executado pelo sistema operativo *Linux*: assim, quanto menos escrevermos, tanto melhor!

Um exemplo comparativo é o que se segue: considere-se que, dentro da árvore de directorias representada na figura **1**, a directoria actual é `/home/estig`, e que a directoria destino é a directoria `textos`, que se encontra dentro da directoria pessoal do mesmo utilizador **eq1234**, que temos vindo a utilizar nestes exemplos. Os caminhos absoluto, relativo e a partir da directoria pessoal são os se seguem:

Absoluto:	`/home/estig/eq/eq1234/textos`
Relativo:	`eq/eq1234/textos`
A partir da directoria pessoal:	`~/textos`

Neste exemplo, é evidente que o caminho que se deve utilizar é o terceiro (a partir da directoria pessoal). Mas nem sempre será assim: haverá muitas outras situações em que um caminho absoluto será mais vantajoso, e também haverá muitas outras situações em que escrever caminhos relativos se revelará mais aconselhável. O que importa realçar aqui é que os caminhos servem para indicar a sequência de directorias que ligam uma directoria actual a uma directoria destino, qualquer que seja o tipo de caminho utilizado.

2.3 Mudança de Directoria

Uma das operações mais utilizadas quando se está a trabalhar com directorias e ficheiros num sistema operativo como o *Linux* é a mudança da directoria actual de trabalho. Como foi discutido no subcapítulo anterior, todas as operações são executadas dentro da directoria actual, a menos que um caminho para outra directoria seja explicitamente referenciado. Portanto, serão muitas as vezes em que se tornará necessário (ou mais cómodo) alterar a directoria considerada como a actual pelo sistema operativo.

Assim, um dos comandos mais utilizados durante uma sessão de trabalho no *Linux* é o comando `cd`, que permite alterar a directoria actual do sistema. O próprio nome "`cd`" é a sigla da expressão inglesa *"change directory"* ("mudar directoria"). A directoria para a qual nos pretendemos posicionar[9] é indicada através de um caminho (absoluto ou relativo). Por exemplo, e considerando a árvore de directorias representada na figura **1**, se a directoria actual for `/home/estig`, e se pretendermos nos posicionar na directoria `/home/estig/eq/eq1234`, então o comando que permite realizar essa tarefa é o seguinte:

```
cd eq/eq1234
```

Todos os vários tipos de caminhos discutidos no subcapítulo anterior podem ser utilizados com o comando `cd`.

[9] Quando é feita uma mudança de directoria, diz-se também que "nos posicionamos" numa determinada directoria.

Se não for indicado qualquer caminho para uma directoria, o comando `cd` simplesmente posiciona o utilizador na sua directoria pessoal. Obter-se-ia o mesmo resultado se fosse utilizado o comando `cd ~`.

No meio das várias mudanças de directoria que poderão ser efectuadas durante uma sessão normal de trabalho, é provável que, momentaneamente, o utilizador não tenha a certeza sobre qual a directoria actual. Para situações como esta, o sistema operativo *Linux* disponibiliza o comando `pwd` (sigla da expressão inglesa *"print working directory"*, ou seja "mostrar directoria de trabalho"). A execução deste comando apresenta o caminho absoluto que leva até à directoria na qual o utilizador se encontra posicionado actualmente.

2.4 Listagem de Ficheiros e Directorias

Para obter uma lista com os ficheiros e subdirectorias[10] existentes numa directoria, utiliza-se o comando `ls`. Se não for indicada a directoria cujo conteúdo se pretende ver, então será apresentada uma lista com o conteúdo da directoria actual.

O comando `ls` possui várias opções que podem ser usadas para alterar o seu comportamento. Se nenhuma opção for utilizada, esse comando apresenta uma lista que contém apenas os nomes dos ficheiros da directoria especificada (se nenhuma for indicada, então será apresentada a lista de ficheiros da directoria actual).

Com a utilização da opção `-a`, o comando `ls` passa a apresentar uma lista como todos os ficheiros da directoria especificada:

```
ls -a
```

Por omissão (isto é, sem a opção `-a`), o comando `ls` não apresenta os ficheiros cujo nome comecem por um ponto ("**.**"). Tais ficheiros são tratados como sendo, de alguma forma, especiais: assim, aparecem "ocultos" aos olhos do utilizador, porque, em geral, são importantes para o funcionamento correcto do sistema operativo, ou da sessão de trabalho do próprio utilizador. Assim, se esses ficheiros estiverem, de alguma maneira, dissimulados, um

[10] A partir deste ponto, sempre que for referido o conteúdo de uma directoria, usar-se-á apenas o conceito de ficheiro em lugar dos dois conceitos de ficheiro e directoria (ou subdirectoria). Como será discutido mais à frente, o sistema operativo *Linux* trata as directorias como ficheiros especiais, pelo que a distinção entre os dois conceitos se revelará irrelevante em muitos pontos da discussão do comando `ls`, e de outros comandos.

utilizador menos experiente não se aperceberá que eles existem, e assim também não lhes poderá fazer "mal" (apagá-los ou alterá-los incorrectamente).

Para que o comando `ls` apresente mais informações sobre os ficheiros contidos no interior da directoria indicada, é necessário executá-lo com a opção `-l`:

```
ls -l
```

Um exemplo da execução do comando anterior é o que se segue:

```
total 20
drwxr-xr-x  5   reis.quarteu  docentes  4096  Feb 25 2002   Desktop
-rw-r--r--  1   reis.quarteu  docentes  334   Jan 3 15:50   directorias
drwx------  2   reis.quarteu  docentes  4096  Nov 20 16:38  mail
drwx------  15  reis.quarteu  docentes  4096  Jan 2 19:15   Maildir
drwxr-xr-x  2   reis.quarteu  docentes  4096  Mar 15 2002   public_html
```

Com a excepção da primeira linha da lista produzida pela execução do comando anterior, cada uma das restantes linhas contém a informação relativa a um ficheiro da directoria especificada no comando. Essa informação está organizada em colunas, cujos significados serão agora explicitados.

A primeira coluna é a mais curiosa de todas, porque consiste numa série de letras "**drwx**" e traços "**-**". Esta coluna descreve as protecções de cada ficheiro e possui o nome técnico de modo. O primeiro caracter do modo identifica o tipo de cada ficheiro, de acordo com a seguinte tabela:

Letra	Tipo de Ficheiro[11]
"`-`"	Um ficheiro normal.
"`d`"	Uma subdirectoria.
"`l`"	Uma ligação (*"link"*), que corresponde a um apontador. Trata-se de um ficheiro que aponta para outro ficheiro[12].

Os dois tipos mais frequentes são o "**-**" e o "**d**", pois os ficheiros normais e as subdirectorias constituem a esmagadora maioria dos ficheiros contidos no sistema operativo. No exemplo que foi apresentado mais acima, pode-se ver que, por exemplo, a segunda linha

[11] Existem mais 4 tipos de ficheiros, os quais estão fora do âmbito desta disciplina: "**c**" (caracter), "**b**" (bloco), "**s**" (*"sockets"*) e "**p**" (*"pipes"*).
[12] Nos sistemas operativos da Microsoft, a partir do *Windows 95*, foi introduzida uma funcionalidade semelhante, cujo nome é atalho (*"shortcut"*).

corresponde a uma subdirectoria (cujo nome é "`Desktop`"), enquanto que a terceira linha corresponde a um ficheiro normal (cujo nome é "`directorias`").

Para além do tipo de ficheiro, o modo possui mais nove caracteres que representam as permissões (ou protecções) associadas a cada ficheiro. As permissões serão analisadas mais detalhadamente no próximo subcapítulo.

A segunda coluna contém números inteiros que representam o número de ligações (*"links"*) associados a cada ficheiro. No caso dos ficheiros, este número costuma ser 1; no caso das directorias, este número está relacionado com o número de subdirectorias contidas no seu interior.

A terceira coluna contém o nome do dono de cada ficheiro, que em geral corresponde ao nome do utilizador que o criou. Neste exemplo, todos os ficheiros da directoria indicada pertencem ao utilizador **reis.quarteu**.

A quarta coluna representa o grupo de utilizadores a que cada ficheiro pertence. Em geral, o grupo do ficheiro é o mesmo grupo ao qual pertence o dono do ficheiro. Neste exemplo, todos os ficheiros pertencem ao grupo **docentes**, o mesmo do utilizador **reis.quarteu**.

A quinta coluna apresenta o tamanho de cada ficheiro, medido em *bytes*. Repare-se que todas as directorias apresentam o mesmo tamanho: 4.096 *bytes* (4 *Kbytes*). Isto ocorre porque o sistema operativo *Linux* trata as directorias como sendo ficheiros especiais, todos com o mesmo tamanho. Nesses ficheiros especiais, o sistema guarda toda a informação necessária para que as directorias saibam sempre qual é o seu conteúdo.

A sexta coluna contém a data em que os ficheiros foram criados ou alterados pela última vez. No caso das directorias, elas são alteradas quando no seu interior é criado ou eliminado algum ficheiro ou subdirectoria.

Por fim, a última coluna corresponde ao nome do ficheiro. Note-se, mais uma vez, que os nomes em *Linux* fazem distinção entre as letras maiúsculas e minúsculas. Assim, seria perfeitamente legal que a directoria aqui em análise tivesse, para além do ficheiro "`directorias`", um outro ficheiro chamado "`Directorias`", já que o sistema operativo não considera que estes nomes sejam iguais: podem, desta forma, representar ficheiros diferentes, dentro de uma única directoria.

É possível executar o comando `ls` (e muitos outros comandos do sistema operativo *Linux*) conjugando várias opções. Por exemplo, se o comando `ls` for executado com as duas opções aqui discutidas (`-l` e `-a`), o resultado será uma lista detalhada com todos os ficheiros existentes dentro da directoria especificada:

```
ls -la
```

A seguir, serão apresentados alguns exemplos de utilização do comando `ls`:

`ls -la .`	Mostra todo o conteúdo da directoria actual.
`ls -la ..`	Mostra todo o conteúdo da directoria-mãe da directoria actual.
`ls -la /`	Mostra todo o conteúdo da directoria raiz.
`ls -la /tmp`	Mostra todo o conteúdo da directoria `tmp`, que se encontra dentro da directoria raiz.
`ls -la ~/musica`	Mostra todo o conteúdo da directoria `musica`, que se encontra dentro da directoria pessoal do utilizador.

2.5 Permissões

2.5.1 Conceitos Gerais

Conforme foi discutido no subcapítulo anterior, a propósito do comando `ls` e da sua opção `-l`, cada ficheiro tem associado uma série de permissões, que determinam os tipos de acesso que podem ser realizados sobre esse ficheiro, e quem pode realizá-los. As permissões são representadas por nove caracteres, que são divididos em três blocos de três caracteres cada. Dentro de cada bloco, um caracter "**r**" significa permissão de leitura (*"read"*); um caracter "**w**" significa permissão de escrita (*"write"*), isto é, alterar, apagar ou renomear; e um caracter "**x**" significa permissão de execução (*"execute"*) dos programas que possam existir nos ficheiros.

No caso das directorias, as permissões têm significados um pouco diferentes. A permissão de leitura ("**r**") refere-se à possibilidade de listar o conteúdo de uma directoria, através da execução do comando `ls`[13]. A permissão de escrita ("**w**") refere-se à possibilidade de se alterar o conteúdo de uma directoria, criando, apagando, movendo ou renomeando

[13] Pode-se considerar que o comando `ls` permite "ler" uma directoria.

ficheiros. A permissão de execução ("**x**") refere-se à possibilidade de executar comandos relacionados com o conteúdo da directoria.

O primeiro bloco de três caracteres representa as permissões associadas ao utilizador (*"user"*) ou dono de cada ficheiro. Sempre que o utilizador cria um novo ficheiro, esse utilizador passa a ser o dono desse ficheiro. O segundo bloco de três caracteres representa as permissões dos utilizadores que pertencem ao mesmo grupo (*"group"*) ao qual o dono do ficheiro também pertence. Finalmente, o terceiro bloco de três caracteres representam as permissões associadas a todos os outros utilizadores (*"others"*) que não se encaixam nas duas situações anteriores, ou seja, ao resto do mundo.

A estrutura das permissões associadas a um ficheiro encontram-se resumidas na tabela seguinte:

Utilizador	Grupo	Outros
rwx	rwx	rwx

No exemplo referente ao comando `ls -l`, do subcapítulo anterior, o ficheiro "`directorias`" possui o conjunto de permissões "`rw-r--r--`". A permissão de leitura ("**r**") está activa nos três blocos de permissões, o que significa que o ficheiro pode ser lido por todos os utilizadores. Pelo contrário, a permissão de escrita ("**w**") só está activa no primeiro bloco, e por essa razão, esse ficheiro só pode ser modificado pelo seu próprio dono. Por fim, a permissão de execução ("**x**") não se encontra activa em nenhum bloco, pelo que este ficheiro não poderá ser executado por ninguém.

No mesmo exemplo, a directoria "`Maildir`" possui as permissões "`rwx------`". Por essa razão, o dono tem permissões para ler, modificar e aceder ao seu interior, mas os restantes utilizadores não têm qualquer acesso ao seu conteúdo. Isto é especialmente importante, porque essa directoria aparentemente guarda as mensagens de correio electrónico enviadas e recebidas pelo utilizador, podendo, assim, conter informação de carácter pessoal.

2.5.2 Alterar Permissões

O comando do sistema operativo *Linux* que se destina à definição e à alteração das permissões de um ou mais ficheiros é o `chmod` (*"change mode"*, que significa "alterar modo"). Como é óbvio, os utilizadores só podem alterar as permissões dos seus próprios

ficheiros. A única excepção a esta regra é o super-utilizador, que tem privilégios especiais para modificar as permissões de qualquer ficheiro do sistema.

O comando `chmod` disponibiliza duas maneiras de definir ou alterar as permissões de um ou mais ficheiros. A primeira maneira é chamada de "modo simbólico", porque utiliza "símbolos" para representar as várias permissões e os vários tipos de utilizadores. Já foi visto anteriormente que as permissões são representadas pelos caracteres "**r**" (leitura), "**w**" (escrita) e "**x**" (execução). Os três tipos de utilizadores são representados da seguinte maneira:

- O dono do ficheiro: "**u**" (de *"user"*);
- Os utilizadores que pertencem ao mesmo grupo que o dono do ficheiro: "**g**" (de *"group"*);
- Os restantes utilizadores do sistema: "**o**" (de *"others"*);
- Todos os utilizadores do sistema: "**a**" (de *"all"*, isto é, "todos").

O modo simbólico do comando `chmod` consiste em dar ou retirar determinadas permissões a determinados tipos de utilizadores. Apenas as permissões especificadas são alteradas para os tipos de utilizadores especificados. Todas as restantes permissões mantém-se inalteradas. Isto será melhor compreendido com a apresentação de alguns exemplos:

`chmod ug+x fich`	Foi atribuída ("+") a permissão de execução ("**x**") do ficheiro "`fich`" ao seu dono ("**u**") e aos utilizadores do grupo ("**g**").
`chmod o-rwx ..`	Todas as permissões ("**rwx**") de acesso à directoria-mãe ("**..**") da directoria actual são retiradas ("**-**") para todos os utilizadores ("**o**") que não sejam o próprio dono daquela directoria, ou que não pertençam ao grupo.
`chmod a-wx fich`	As permissões de escrita ("**w**") e execução ("**x**") sobre o ficheiro "`fich`" foram retiradas ("**-**") a todos os utilizadores ("**a**")[14].
`chmod u+rwx,g=rx,o-wx .`	O dono ("**u**") da directoria actual ("**.**") passa a ter ("+") acesso total ("**rwx**") sobre essa directoria. Os utilizadores do grupo ("**g**") apenas têm ("=") permissões de leitura ("**r**") e de execução ("**x**") sobre essa directoria. Em relação aos restantes utilizadores ("**o**"), foram removidas ("**-**") as permissões de escrita ("**w**") e de execução ("**x**").

[14] Quando se pretende atribuir ou retirar as mesmas permissões a todos os utilizadores, não é necessário explicitar o tipo de utilizador ("**a**"). Este comando poder-se-ia escrever assim: `chmod -wx fich`.

Em resumo, o modo simbólico do comando `chmod` pode ser representado de acordo com a seguinte sintaxe:

```
chmod [ugoa][+-=][rwx] nome_ficheiro
```

O comando `chmod` dispõe de um segundo modo de execução: o "modo absoluto". Neste modo, todas as permissões de todos os tipos de utilizadores são indicadas de uma única vez, utilizando uma representação numérica. Como foi discutido anteriormente, as nove permissões são divididas em três grupos para cada tipo de utilizador. Para cada uma das permissões, deve-se substituir a letra que representa a permissão ("**r**", "**w**" ou "**x**") pelo dígito binário (*"bit"*) 1, e deve-se substituir o traço ("**-**") pelo dígito binário 0. Desta forma, obtém-se uma representação binária para cada grupo de três permissões, uma para cada tipo de utilizador. Substitui-se cada grupo de três *bits* pelo seu correspondente dígito octal (isto é, de base numérica 8), obtendo-se a representação octal das permissões que se pretende definir para um ou mais ficheiros. É essa representação octal que será utilizada no comando `chmod`.

A equivalência entre os números octais e as permissões é dada pela tabela que se segue:

Permissões	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`

Seguem-se alguns exemplos da utilização do comando `chmod` em modo absoluto:

`chmod 754 .`	O dono da directoria actual ("**.**") passa a ter acesso total (**7**) sobre essa directoria. Os utilizadores do grupo apenas têm permissões de leitura e de execução (**5**) sobre essa directoria. Os restantes utilizadores apenas podem ler (**4**) essa directoria.
`chmod 740 fich`	O dono do ficheiro "`fich`" passa a ter todas as permissões (**7**) de acesso a esse ficheiro. Os utilizadores do grupo apenas têm permissão de leitura (**4**). Os restantes utilizadores não têm quaisquer permissões (**0**) sobre esse ficheiro.

As pessoas que conhecem bem os sistemas de numeração binário, octal e hexadecimal (base 16) costumam preferir o modo absoluto do comando `chmod`, porque é mais compacto. Contudo, as pessoas que não estão familiarizadas com este tipo de numeração podem continuar a usar o modo simbólico, porque também permite utilizar todas as operações de alteração e definição de permissões.

2.5.3 Alterar Dono e Grupo

Para além do comando `chmod`, o sistema operativo *Linux* disponibiliza mais dois comandos que estão relacionados com as permissões dos ficheiros:

- O comando `chown` (*"change owner"*), que serve para modificar o dono do ficheiro;
- O comando `chgrp` (*"change group"*), que serve para modificar o grupo do ficheiro.

Como sempre, os utilizadores normais (ou seja, os que não são o super-utilizador) só possuem autorização para alterar o dono dos seus próprios ficheiros, ou seja, para oferecerem os seus ficheiros a outros utilizadores.

Um exemplo de utilização do comando `chown` é o que se segue:

```
chown novo_dono ~/fich
```

Aqui, o ficheiro "`fich`", localizado na directoria pessoal do utilizador actual, é atribuído ao utilizador "`novo_dono`", que passa assim a ser o seu novo dono. A partir desse momento, o dono anterior não poderá desfazer esta operação, uma vez que deixou de ser o dono do ficheiro (e apenas os donos dos ficheiros podem oferecê-los a outros utilizadores).

Um exemplo de utilização do comando `chgrp` é o que se segue:

```
chgrp novo_grupo ~/fich
```

A execução deste comando atribui o ficheiro "`fich`", localizado na directoria pessoal do utilizador actual, ao grupo de utilizadores "`novo_grupo`". Ao contrário do exemplo anterior, aqui o dono do ficheiro continua a ser o mesmo. Por essa razão, o dono pode voltar a modificar as permissões do ficheiro, incluindo o grupo, o dono e o modo.

2.6 Caracteres Genéricos

No trabalho do dia a dia, é muito frequente sermos confrontados com a necessidade de realizar a mesma tarefa sobre múltiplos ficheiros. Para evitar ter de repetir o mesmo comando muitas vezes, existem os caracteres genéricos (*"wildcards"*), isto é, caracteres especiais, usados para definir, com um único nome, listas de ficheiros. Os caracteres genéricos são os seguintes:

Caracter	Significado
`*`	Qualquer sequência de zero ou mais caracteres.
`?`	Um e um só caracter, qualquer que ele seja.
`[ ]`	Uma sequência de um ou mais caracteres pertencentes ao conjunto definido entre os parêntesis rectos.

Vamos examinar uma série de exemplos para uma melhor compreensão dos caracteres genéricos.

Comando	Significado
`ls -l ~/p*`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com o caracter "**p**" (ex.: `p`, `p1`, `pessoal`, `peixe`).
`ls -l ~/p*.o`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com o caracter "**p**" e terminem com os caracteres "**.o**" (ex.: `p.o`, `programa.o`).
`ls -l ~/program?.o`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com os caracteres "**program**", seguidos de um único qualquer caracter, e terminem com os caracteres "**.o**" (ex.: `programa.o`, `programb.o`, `program2.o`).
`ls -l ~/program?*.o`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com os caracteres "**program**", seguidos de um ou mais caracteres, e terminem com os caracteres "**.o**" (ex.: `programa.o`, `programbasic.o`, `program23.o`).
`ls -l ~/program[a-c]*`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com os caracteres "**program**", seguidos de um único caracter da gama compreendida entre os caracteres "**a**" e "**c**", inclusive, seguidos de um grupo de zero ou mais caracteres (ex.: `programa.o`, `programbasic`, `programc.pas`).
`ls -l ~/program[a-c]*.o`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com os caracteres "**program**", seguidos de um único caracter da gama compreendida entre os caracteres "**a**" e "**c**", inclusive, seguidos de um grupo de zero ou mais caracteres, e terminem com os caracteres "**.o**" (ex.: `programa.o`, `programbasic.o`, `programc23.o`).

`ls -l ~/program[ac]*.o`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com os caracteres "**program**", seguidos ou do caracter "**a**" ou do caracter "**c**", seguidos de um grupo de zero ou mais caracteres, e terminem com os caracteres "**.o**" (ex.: `programa.o`, `programcobol.o`, `programa_isi.o`).
`ls -l ~/program[!ac]*.o`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com os caracteres "**program**", seguidos de um único caracter que seja diferente de "**a**" e de "**c**", seguidos de um grupo de zero ou mais caracteres, e terminem com os caracteres "**.o**" (ex.: `programbasico.o`, `program_cobol.o`).
`ls -la ~/.*`	Lista todos os ficheiros da directoria pessoal cujos nomes comecem com um ponto ("**.**").
`ls -la ~/*x*`	Lista todos os ficheiros da directoria pessoal cujos nomes tenham pelo menos um caracter "**x**" (ex.: `exemplo`, `x`, `xefe.cpp`, `music.mix`).
`ls -la ~/?x*`	Lista todos os ficheiros da directoria pessoal cujo segundo caracter do nome seja "**x**" (ex.: exemplo, `exercicio`).
`ls -la ~/???`	Lista todos os ficheiros da directoria pessoal cujo nomes tenham um tamanho de exactamente três caracteres (ex.: `ash`, `sed`, `red`, `bsh`, `awk`, `pwd`, `cat`)

Os caracteres genéricos podem ser utilizados em conjugação com qualquer comando que utilize ficheiros ou directorias.

2.7 Gestão de Ficheiros

2.7.1 Copiar Ficheiros

Para copiar ficheiros de uma directoria para outra, o sistema operativo *Linux* disponibiliza o comando `cp` (*"copy"*). Este comando também pode ser utilizado para copiar ficheiros dentro da mesma directoria, desde que sejam usados nomes diferentes para o(s) ficheiro(s) original(is) e para a(s) cópia(s). Por esse motivo, o comando `cp` necessita sempre de dois nomes: o nome do(s) ficheiro(s) original(is) e o nome do(s) novo(s) ficheiro(s). Um exemplo deste comando é o seguinte:

```
cp /etc/passwd ~/abcd
```

Neste exemplo, o ficheiro "`passwd`", existente na subdirectoria "`etc`" da directoria raiz, é copiado para o ficheiro "`abcd`" da directoria pessoal do actual utilizador. Se o ficheiro

"abcd" não existir, é criado então um ficheiro com esse nome, que passará a ser uma cópia do ficheiro "passwd" original. Pelo contrário, se o ficheiro "abcd" existir, então o seu conteúdo actual será destruído e completamente substituído pelo conteúdo do ficheiro "passwd".

Por outro lado, se o ficheiro "abcd" for uma directoria[15], então o ficheiro "passwd" é copiado para dentro dessa directoria: o novo ficheiro terá o nome do ficheiro original, "passwd". Assim, quando se deseja fazer uma cópia de um ficheiro localizado numa determinada directoria para outra directoria, mantendo o nome original do ficheiro, basta usar o nome desta última directoria como o destino do ficheiro original.

Outro exemplo do comando cp:

```
cp origem destino
```

Aqui, o ficheiro "origem", contido na directoria actual, é copiado para a própria directoria actual, mas com outro nome: "destino". Assim, dentro da directoria actual passam a existir dois ficheiros exactamente iguais, com a excepção dos seus nomes.

O comando cp também possui várias opções que alteram o seu funcionamento normal. Veja-se este exemplo:

```
cp -Rv /tmp/isi ~
```

Este exemplo apresenta duas opções do comando cp. A opção -R (*"recursive"*) permite copiar não só os ficheiros que se encontram na directoria /tmp/isi, mas também todas as subdirectorias e respectivos ficheiros dessa directoria. Desta forma, toda a estrutura da directoria /tmp/isi é copiada para a directoria pessoal do actual utilizador.

A opção -v (*"verbose"*) produz, para cada ficheiro que seja copiado, uma linha informativa com o nome e a localização do ficheiro original, acompanhados pelo nome e pela localização do ficheiro copiado.

Os ficheiros criados através do comando cp mantém todas as permissões dos ficheiros originais. Contudo, os ficheiros copiados deixam de pertencem ao dono original e passam a pertencer ao utilizador que fez a cópia. Por outro lado, para que um utilizador possa copiar

[15] Lembre-se de que o sistema operativo *Linux* trata as directorias como ficheiros com significado especial.

ficheiros para dentro de uma directoria, é necessário que ele possua permissão de escrita e de execução sobre essa directoria.

2.7.2 Mover Ficheiros

Para mover ou alterar o nome de um ficheiro, temos o comando `mv` (*"move"*). Tal como o comando `cp`, também o comando `mv` necessita de dois nomes: o nome do(s) ficheiro (s) original(is) e o nome do(s) ficheiro(s) destino(s). Quando um ou mais ficheiros são movidos de uma directoria para outra, cada ficheiro movido dá origem a uma cópia na directoria de destino, enquanto que todos os ficheiros originais são apagados. Esta é uma diferença importante entre os comandos `cp` e `mv`. Um exemplo deste comando é o seguinte:

```
mv ~/programas/prog1.c ~/trabalho
```

Neste exemplo, o ficheiro "`prog1.c`", existente na subdirectoria "`programas`" da directoria pessoal do utilizador actual, é movido para o ficheiro "`trabalho`" da directoria pessoal do mesmo utilizador. Se o ficheiro "`trabalho`" não existir, é criado então um ficheiro com esse nome, que passará a ser uma cópia do ficheiro "`prog1.c`" original. Este, por sua vez, é automaticamente eliminado da directoria "`programas`". Por outro lado, se o ficheiro "`trabalho`" existir, então o seu actual conteúdo será destruído e completamente substituído pelo conteúdo do ficheiro "`prog1.c`".

Por outro lado, se o ficheiro "`trabalho`" for, na verdade, uma directoria, então o ficheiro "`prog1.c`" é movido para dentro dessa directoria, mantendo o nome do ficheiro original, "`prog1.c`". Assim, quando se deseja mover um ficheiro localizado numa determinada directoria para outra directoria, mantendo o nome original, basta usar o nome desta última directoria como o destino do ficheiro original.

Outro exemplo do comando `mv`:

```
mv README readme
```

Aqui, o ficheiro "`README`", contido na directoria actual, é movido para a própria directoria actual, mas com outro nome: "`readme`". Na verdade, esta operação corresponde apenas à alteração do nome (renomeação) do ficheiro "`README`" para "`readme`".

O comando `mv` pode também ser aplicado directamente sobre directorias, não necessitando, assim, de uma opção `-R` como o comando `cp`. Quando se move uma directoria, todos os ficheiros que se encontram no seu interior são automaticamente movidos.

Os ficheiros movidos através do comando `mv` mantém todas as permissões dos ficheiros originais. Contudo, nem todos os ficheiros podem ser movidos: é necessário que o utilizador tenha permissão de escrita e de execução sobre a directoria em que se encontra o ficheiro que se pretende mover e sobre a directoria para onde se pretende mover o ficheiro.

2.7.3 Apagar Ficheiros

Para além de copiar e mover ficheiros, também é necessário, muitas vezes, eliminar ficheiros, a fim de libertar espaço em disco. O comando *Linux* usado para remover ficheiros é o `rm` (*"remove"*). Por exemplo, para eliminar o ficheiro "`calendario`", localizado na directoria actual do sistema, basta executar o comando que se segue:

```
rm calendario
```

Por norma, o comando `rm` só funciona sobre ficheiros. Para remover directorias, deve ser utilizado a opção `-r`, que remove árvores de directorias recursivamente[16]. Por exemplo, para remover todo o conteúdo (ficheiros e subdirectorias) da directoria "`subdir`", localizada na directoria pessoal do actual utilizador, basta executar o seguinte comando:

```
rm -r subdir
```

O comando `rm`, em conjunto com a opção `-r`, é bastante útil em situações onde existem estruturas de directorias complicadas, que necessitam de ser eliminadas. Contudo, é necessário ter muito cuidado antes de o utilizar, uma vez que os seus efeitos podem ser devastadores. Quando não estiver absolutamente seguro do que está a fazer ao utilizar o comando `rm` com a opção `-r`, então deverá utilizar também a opção `-i`. Assim, todo e qualquer ficheiro que venha a ser eliminado terá de ter a confirmação do utilizador.

```
rm -ri subdir
```

Para se apagar um ficheiro, é necessário que o utilizador tenha permissão de escrita sobre esse ficheiro. Se não for este o caso, o sistema pede ao utilizador que confirme que

[16] Também se pode utilizar o comando `rmdir`, que será discutido mais à frente.

realmente pretende apagar um ficheiro protegido contra a escrita. Além disso, também é necessário que o utilizador tenha permissão de escrita e de execução sobre a directoria na qual se encontra o ficheiro que se quer apagar. Caso contrário, o sistema operativo não permitirá que o ficheiro seja eliminado.

2.8 Gestão de Directorias

2.8.1 Criar Directorias

O comando `mkdir` (*"make directory"*) cria uma nova directoria dentro da directoria actual, ou dentro de outra directoria cujo nome seja especificado na linha de comando. Por exemplo, para que uma nova directoria, cujo nome é "`cursos`", seja criada dentro da directoria pessoal do utilizador actual, deve-se executar o seguinte comando:

```
mkdir ~/cursos
```

Para que uma nova directoria possa ser criada, é necessário que o utilizador possua permissão de escrita e de execução sobre a directoria na qual a nova directoria será criada.

2.8.2 Apagar Directorias

O comando `rmdir` (*"remove directory"*) serve para remover uma ou mais directorias. Por exemplo, para que a directoria "`cursos`" seja eliminada da directoria pessoal do utilizador actual, deve-se executar o seguinte comando:

```
rmdir ~/cursos
```

Para que uma directoria possa ser apagada, a primeira condição que deve ser respeitada é que a directoria esteja completamente vazia: não pode existir nada no seu interior, nem ficheiros, nem subdirectorias[17]. Por outro lado, também é necessário que o utilizador possua permissão de escrita e de execução sobre a directoria-mãe da directoria que se quer apagar. Retornando ao exemplo anterior, o utilizador precisa de ter permissão de escrita e de execução sobre a sua directoria pessoal para que a directoria "`cursos`" possa ser apagada.

[17] É por esta razão que se costuma utilizar o comando `rm -r` para eliminar directorias com estruturas complexas.

2.9 Visualização de Ficheiros

No que se refere à visualização de ficheiros de texto, o sistema operativo *Linux* oferece os comandos `cat` e `more`[18]. O comando `cat`[19] mostra simplesmente o conteúdo do ficheiro especificado. Por exemplo, para visualizar o conteúdo de um ficheiro chamado "`readme`", existente na directoria actual, executa-se o comando seguinte:

```
cat readme
```

Quando os ficheiros são muito extensos, as listagens produzidas pelo comando `cat` podem não caber integralmente no ecrã do computador. Nestes casos, deve-se utilizar os comandos `more` (ou `less`), que apresentam listagens com pausas no final de cada página. Por exemplo, para ver o conteúdo do ficheiro `/etc/services`, com paragens página a página, basta usar o seguinte comando:

```
more /etc/services
```

A apresentação de ficheiros através do comando `more` é feita com a utilização das seguintes teclas:

Tecla	Significado
`Barra de Espaços`	Passar para a página seguinte.
`Enter`	Passar para a linha seguinte.
`/`	Pesquisar palavras dentro do ficheiro, saltando directamente para a próxima ocorrência da palavra procurada.
`q`	Parar a listagem, e terminar a execução do comando.

Existem mais dois comandos que são muito úteis para o tratamento de ficheiros extensos: `head` e `tail`. Por omissão, o comando `head` apresenta apenas as primeiras 10 linhas do ficheiro especificado, enquanto que o comando `tail` apresenta apenas as últimas 10 linhas do ficheiro especificado. É possível alterar o número de linhas apresentadas por cada comando, usando a opção `-n`, seguida do número de linhas que se pretende visualizar. Vejam-se estes exemplos:

[18] Existe também um comando chamado `less`, que é uma variante muito melhorada do comando `more`, ao contrário do que o seu nome possa sugerir.
[19] O nome deste comando não tem nada a ver com o animal felino mais conhecido por gato (*"cat"*, em inglês), mas sim com uma contracção da palavra inglesa *"concatenate"* ("concatenar", ou seja, juntar duas ou mais cadeias alfanuméricas).

Comando	Significado
`head readme`	Apresenta as 10 primeiras linhas do ficheiro "`readme`".
`head -n 2 readme`	Apresenta as 2 primeiras linhas do ficheiro "`readme` ".
`tail readme`	Apresenta as 10 últimas linhas do ficheiro "`readme` ".
`tail -n 5 readme`	Apresenta as 5 últimas linhas do ficheiro "`readme` ".

2.10 Pesquisa de Ficheiros

O comando mais utilizado em *Linux* para procurar informação é o `find`. Este é um comando bastante poderoso e versátil, porque possui dezenas de opções. Por exemplo, o comando que se segue permite procurar um ficheiro denominado "`ghostscript`" dentro da directoria `/usr`:

```
find /usr -name "ghostscript"
```

O comando anterior pode ser interpretado da seguinte maneira: pretende-se procurar, dentro da directoria `/usr`, um ficheiro cujo nome é "`ghostscript`". Para além do nome, também é possível pesquisar ficheiros utilizando muitos outros critérios: pela data da última modificação, pelo tamanho, pelas permissões, pelo nome do dono ou do grupo, etc. Todos estes critérios podem também ser conjugados com os operadores lógicos *and*, *or* e *not*.

Outro exemplo de utilização do comando `find`: para pesquisar dentro da directoria `/tmp` todos os ficheiros cujos nomes terminem com a expressão "`.o`", deve-se executar o comando que se segue:

```
find /tmp -name "*.o"
```

3. Redireccionamento

3.1 Redireccionamento da saída de dados de um comando

Qualquer comando ou programa que seja executado no sistema operativo *Linux* está sempre associado a três ficheiros virtuais de entrada e saída de dados. Esses ficheiros virtuais denominam-se: `stdin` (*"standard input"*, ou "entrada padrão"), `stdout` (*"standard output"*, ou "saída padrão") e `stderr` (*"standard error"*, ou "erro padrão"). O `stdin` é utilizado pelos programas para ler informação. O segundo ficheiro, `stdout`, é um ficheiro virtual para onde os programas enviam toda a informação que produzem. Finalmente, o terceiro ficheiro, `stderr`, é para onde são enviadas as mensagens de erro produzidas pelos programas.

Por norma, o ficheiro virtual `stdin` está redireccionado para o teclado do utilizador, enquanto que os ficheiros virtuais `stdout` e `stderr` estão redireccionados para o monitor.

Por exemplo, quando o comando "`ls -l`" é executado, o programa `ls` limita-se a escrever a listagem da directoria actual para o ficheiro virtual `stdout`. Como este ficheiro está redireccionado para o monitor, todo o texto que o comando `ls` escreve vai aparecer no monitor.

Contudo, é possível redireccionar qualquer dos três ficheiros virtuais para outro lado. Por exemplo, veja-se o seguinte comando:

```
ls -l > listagem.txt
```

Neste exemplo, o caracter "**>**" é usado para redireccionar o ficheiro virtual `stdout` (ou seja, a saída do comando "`ls -l`") para um ficheiro chamado "`listagem.txt`", contido na directoria actual do sistema operativo. Desta forma, a listagem da directoria não aparece no monitor. Pelo contrário, um novo ficheiro chamado "`listagem.txt`" é criado e os resultados produzidos pela execução do comando "`ls -l`" são gravados nesse ficheiro. Para visualizar a listagem produzida pelo comando, deve-se visualizar o conteúdo do ficheiro "`listagem.txt`", utilizando um dos comandos de visualização de ficheiros de texto (`cat`, `more` ou `less`).

O caracter de redireccionamento da saída, "**>**", cria sempre um novo ficheiro, usando o nome que é indicado no comando. Se já existir um ficheiro com esse nome, então o seu

conteúdo é destruído, sendo completamente substituído pelo resultado da execução do comando. Para evitar que isso aconteça, deve-se utilizar os caracteres de redireccionamento da saída ">>". Assim, o ficheiro indicado apenas é criado se não existir. Se o ficheiro já existir, o resultado da execução do comando é enviado para o fim desse ficheiro, preservando assim o seu conteúdo original. Por exemplo:

```
ls -l >> listagem.txt
```

3.2 Redireccionamento da entrada de dados de um comando

É possível também redireccionar a entrada de dados de um comando: para isso, basta utilizar o caracter "<" para que o programa vá buscar os dados de que precisa no ficheiro cujo nome é indicado à direita daquele caracter. Por exemplo, o próximo comando copia toda a informação contida no ficheiro "`listagem.txt`" para outro ficheiro chamado "`listagem2.txt`":

```
cat < listagem.txt > listagem2.txt
```

Neste exemplo, o comando `cat` limita-se a ler o ficheiro de entrada, que está redireccionado para o ficheiro "`listagem.txt`", e a apresentar o conteúdo desse ficheiro na saída, que está redireccionada para o ficheiro "`listagem.txt`". O resultado é uma cópia exacta[20].

3.3 Comunicação entre programas através de redireccionamento

O redireccionamento do sistema operativo *Linux* permite também que o resultado da execução de um comando seja directamente enviado para a entrada de dados de outro comando. Isto consegue-se com a utilização do caracter "|", mais conhecido pela sua designação inglesa *"pipe"* ("tubo"), que estabelece um canal de comunicação entre dois programas[21]. Por exemplo:

```
ls -l /usr/bin | more
```

[20] O comando do exemplo anterior é equivalente ao comando "`cp listagem.txt listagem2.txt`". É apresentado apenas por razões pedagógicas.
[21] Esse "canal" de comunicação é visto como um "tubo" por onde a informação produzida por um programa passa até chegar ao outro programa.

Neste exemplo, o comando "`ls -l /usr/bin`" lista o conteúdo da directoria `/usr/bin`, que contém aproximadamente dois mil ficheiros. O efeito do "tubo" consiste em redireccionar a saída do comando `ls` para a entrada do comando `more`. Como resultado, consegue-se obter uma listagem de directoria com pausas no fim de cada página.

Em *Linux*, os "tubos" são usados com muita frequência, porque permitem conjugar o efeito de vários comandos muito simples para realizar tarefas de grande complexidade.

3.4 Redireccionamento de erros

Por fim, o redireccionamento da saída de erros é feita através dos caracteres "**>&**". Estes caracteres redireccionam não apenas a saída de erros, mas também a saída de dados. Por exemplo, quando foi aqui discutido o comando `find`, foram apresentados dois exemplos que, se forem executados, geram muitos erros. Esses erros têm origem na tentativa que se faz para pesquisar o conteúdo de directorias sobre as quais não se tem permissão de leitura e/ou de execução. Vamos, então, redireccionar não só os resultados produzidos por um comando `find`, mas também os seus erros:

```
find /tmp/isi -name "*.c" &> resultado
```

Neste exemplo, pretende-se procurar, dentro da directoria `/tmp/isi`, ficheiros cujos nomes terminem com a expressão "`.c`". O resultado desta procura (que deveria ser enviado para o ficheiro virtual `stdout`) e os eventuais erros gerados por essa procura (que deveriam ser enviados para o ficheiro virtual `stderr`) são ambos redireccionados para um ficheiro chamado "`resultado`", localizado na directoria actual do sistema.

4. Processamento de Texto

4.1 Selecção de Linhas de Texto

O sistema operativo *Linux* possui vários utilitários para o processamento de ficheiros de texto, nomeadamente para a selecção de texto que obedeça a determinadas condições, e para a transformação de texto de um formato para outro. De entre esses utilitários, provavelmente o mais poderoso de todos é o comando `grep`. Este comando selecciona as linhas de texto, dos ficheiros especificados, que respeitam um determinado padrão. Por exemplo, para seleccionar as linhas de texto, de um ficheiro chamado "`FAQ`", que contenham a palavra (ou padrão) "risk", bastaria escrever o seguinte comando:

```
grep risk FAQ
```

Em seguida, serão apresentados vários exemplos de utilização do comando `grep`, ao lado dos seus respectivos significados:

Comandos	Significados
`grep -c risk FAQ`	Calcula o número de linhas do ficheiro `FAQ` em que o padrão "risk" ocorre.
`grep -n risk FAQ`	Selecciona todas as linhas do ficheiro `FAQ` em que o padrão "risk" ocorre, antecedidas do seu número dentro do ficheiro.
`grep -v risk FAQ`	Selecciona todas as linhas do ficheiro `FAQ` em que o padrão "risk" não ocorre.
`ls -la | grep lixo`	O comando `ls` produz a informação de todos os ficheiros da directoria actual, e o comando `grep` selecciona apenas as linhas onde ocorre o padrão "lixo"[22].
`ls -la | grep -v lixo`	O comando `ls` produz a informação de todos os ficheiros da directoria actual, e o comando `grep` selecciona apenas as linhas onde não ocorre o padrão "lixo".
`ls -la | grep lixo | head -n 1`	O comando `ls` produz a informação de todos os ficheiros da directoria actual, o comando `grep` selecciona apenas as linhas onde ocorre o padrão "lixo", e o comando `head` apresenta a primeira das linhas seleccionadas.
`cat FAQ | grep risk`	Equivalente ao comando "`grep risk FAQ`".

[22] Admitindo que nenhum utilizador ou grupo de utilizadores terá o nome de "lixo", então o padrão refere-se ao nome dos ficheiros. Assim, esse comando é equivalente ao comando `ls -la *lixo*`.

`cat FAQ | grep you -i | more`	Selecciona todas as linhas do ficheiro `FAQ` em que o padrão "you" ocorre, não fazendo distinção entre as letras maiúsculas e minúsculas (opção `-i`), e apresenta-as página a página (comando `more`).

4.2 Ordenação de Texto

O comando `sort` é usado, tal como o seu nome indica, para ordenar as linhas de um ficheiro de texto por uma determinada ordem, indicada pelo utilizador. Existem várias opções que permitem definir a forma como o texto é ordenado: ordem alfabética, ordem numérica, crescente ou decrescente, etc.

Para melhor ilustrar o funcionamento do comando `sort`, considere-se um ficheiro de texto, cujo nome é "`dicionario`", e cujo conteúdo é formado pelas linhas de texto seguintes:

```
cao animal
vaca animal
gato animal
gato animal
rosa flor
tulipa flor
laranja fruta
```

O resultado da aplicação do comando `sort` ao ficheiro "`dicionario`" seria a ordenação das linhas do ficheiro por ordem alfabética crescente: esta é a ordenação que o comando `sort` utiliza por omissão. Assim, a execução do comando:

```
sort dicionario
```

Produziria o seguinte resultado:

```
cao animal
gato animal
gato animal
laranja fruta
rosa flor
tulipa flor
vaca animal
```

Para eliminar linhas repetidas no resultado, deve-se utilizar a opção `-u` do comando `sort`. Assim, o comando de ordenação passaria a ser o seguinte:

```
sort -u dicionario
```

Este comando produziria o seguinte resultado:

```
cao animal
gato animal
laranja fruta
rosa flor
tulipa flor
vaca animal
```

Para que apenas as linhas referentes aos animais aparecessem ordenadas por ordem alfabética crescente, então seria necessário conjugar a saída padrão do comando `sort` com a entrada padrão do comando `grep`, através de um "tubo". O comando resultante seria:

```
sort dicionario | grep animal
```

O resultado da execução deste comando seria o seguinte:

```
cao animal
gato animal
gato animal
vaca animal
```

4.3 Estatísticas de Ficheiros de Texto

Para o cálculo de estatísticas relacionadas com ficheiros de texto, o sistema operativo *Linux* disponibiliza o comando `wc` (*"word count"*). As estatísticas produzidas por este utilitário são o número de linhas, o número de palavras e o número total de caracteres existentes em cada ficheiro. Para se obter aquelas estatísticas, deve-se utilizar o comando `wc` com as suas opções `-l`, `-w` e `-c`, respectivamente. Alguns exemplos da sua utilização em conjugação com outros comandos de processamento de texto são os seguintes:

Comandos	Significados
`sort -u dicionario | grep animal | wc -l`[23]	Conta quantas linhas não repetidas do ficheiro "`dicionario`" contém a palavra "animal"[24].
`grep -v animal dicionario | wc -l`[25]	Conta quantas linhas do ficheiro "`dicionario`" não contém a palavra "animal".

4.4 Selecção de Colunas de Texto

Finalmente, o comando `cut` serve para partir um ficheiro de texto em várias colunas. Ou seja, em vez de listar todo o conteúdo de um ficheiro, este comando filtra apenas uma determinada gama de caracteres de cada linha.

No exemplo seguinte, considere-se um ficheiro de texto chamado "`palavras`". O comando `cut` será usado para partir esse ficheiro em três colunas, que depois são redireccionadas para três ficheiros diferentes:

```
cut -b 1-10 palavras > primeira_coluna
cut -b 11-20 palavras > segunda_coluna
cut -b 21- palavras > terceira_coluna
```

[23] O comando `wc` poderia ser aqui substituído pela opção `-c` do comando `grep`, uma vez que esta opção calcula o número de linhas em que ocorre o padrão especificado. Assim, o comando ficaria assim: `sort -u dicionario | grep -c animal`.
[24] O ficheiro "dicionario" é ordenado por ordem alfabética crescente, sendo-lhe retiradas as linhas repetidas. Depois, são seleccionadas e contadas apenas as linhas que contenham a palavra "animal".
[25] Aqui, o comando `wc` também poderia ser substituído pela opção `-c` do comando `grep`. Assim, o comando ficaria assim: `grep -c -v animal dicionario`.

O primeiro comando deste exemplo selecciona os primeiros dez caracteres (colunas 1 a 10) do ficheiro "palavras". É necessário notar que o ficheiro "palavras" não é modificado: o comando cut limita-se a enviar o resultado da filtragem para a saída padrão[26]. No segundo comando, são filtrados os 10 caracteres seguintes, ou seja, os caracteres que se encontram entre a coluna 11 e a coluna 20, inclusive. Finalmente, no terceiro comando, são filtrados todos os caracteres que vão desde a coluna 21 até ao fim da linha.

O comando cut também é capaz de dividir um ficheiro de texto em colunas baseadas não em posições, como no exemplo anterior, mas sim em caracteres delimitadores de campos[27]. Para isso, deve-se usar duas outras opções desse comando:

- A opção -d, que indica qual o caracter delimitador que definirá as colunas do ficheiro de texto;
- A opção -f, que indica qual a coluna (definida pelo caracter delimitador) do ficheiro de texto que será extraída.

Voltando a considerar o ficheiro "dicionario", que foi usado para ilustrar alguns exemplos anteriores, vamos dividi-lo em colunas em que o caracter delimitador seja o espaço em branco:

```
cut -d ' ' -f 1 dicionario
```

Este comando extrai a primeira coluna de palavras (-f 1) do ficheiro "dicionario", sendo o espaço em branco o caracter delimitador (-d ' '). O resultado da execução deste comando é o seguinte:

```
cao
vaca
gato
gato
rosa
tulipa
laranja
```

[26] O mesmo é feito pelos outros comandos de processamento de texto (grep, sort e wc) estudados neste capítulo: nenhum deles modifica os ficheiros que lhes fornecem os dados necessários ao seu funcionamento.
[27] Exactamente como numa tabela de uma base de dados: cada campo de uma tabela está separado do campo seguinte por um caracter delimitador.

Para determinar quantas categorias de seres vivos existem no ficheiro "`dicionario`", poder-se-ia executar o comando que se segue:

```
cut –d ' ' –f 2 dicionario | sort –u | wc –l
```

Usando o espaço em branco como caracter delimitador, é extraída a segunda coluna do ficheiro "`dicionario`". Essa coluna é ordenada e as linhas repetidas são retiradas. Isso permite que o resultado da ordenação possa ser enviado para o comando `wc` para que o número de linhas possa ser contado. Esse número é o número total de categorias (animal, flor e fruta) que se pretendia calcular.

O comando `cut` é especialmente útil para extrair colunas de informação produzidas por muitos comandos *Linux*. Por exemplo, para se saber quantos utilizadores estão ligados agora à mesma máquina em nos encontramos a trabalhar, apenas se torna necessário ligar o comando `who` aos comandos encarregados de filtrar e contar essa informação:

```
who | cut –d ' ' –f 1 | sort –u | wc –l
```

O comando `who` produz informação tabelada sobre os utilizadores que se encontram actualmente a trabalhar. O comando `cut` extrai a primeira coluna dessa informação: essa é a coluna com os nomes dos utilizadores. O comando `sort` ordena esses nomes e retira quaisquer nomes duplicados que possam estar presentes. Isso garante que o comando `wc` apenas contará cada nome de utilizador uma única vez.

5. O Interpretador de Comandos

O interpretador de comandos é o programa responsável pela interpretação e execução dos comandos introduzidos pelo utilizador durante uma sessão de trabalho no sistema operativo *Linux*. O nome que tradicionalmente se dá ao interpretador de comandos dos sistemas *Linux* é *"shell"* ("concha", em inglês), no sentido que este programa "esconde" as complexidades internas do funcionamento do sistema operativo (da mesma forma que uma concha esconde o seu interior dos olhares externos).

5.1 Ficheiros Especiais de Configuração

Existem vários ficheiros especiais que permitem configurar tanto o interpretador de comandos como o ambiente de trabalho de uma sessão. Esses ficheiros de texto são constituídos por sequências de comandos que são interpretados e executados de uma só vez, como se o ficheiro fosse um comando. Desta forma, o utilizador pode escrever nesses ficheiros todos os comandos que pretenda que sejam executados sempre que determinadas condições ocorram.

5.1.1 O Ficheiro .bash_profile

O primeiro desses ficheiros especiais de configuração tem o nome de `.bash_profile`, que é executado sempre que uma nova sessão de trabalho é iniciada, ou seja, sempre que um utilizador entra no sistema. Para ver o seu conteúdo, basta visualizá-lo através do comando `cat`.

Para alterar o seu conteúdo, acrescentando-lhe novos comandos, retirando-lhe ou alterando-lhe comandos já existentes, pode-se utilizar um dos processadores de texto disponibilizados pelo *Linux*, o `pico`. A sua utilização é bastante intuitiva e facilitada pela presença, na parte inferior do monitor de uma barra com as combinações de teclas[28] que se devem utilizar para efectuar determinadas operações.

```
pico .bash_profile
```

[28] Na barra de ajuda do `pico`, o caracter "^" significa que se deve carregar na tecla **Ctrl** em simultâneo com a tecla que aparece representada à direita de "^". Por exemplo, para obter ajuda sobre o funcionamento do `pico`, deverá carregar a combinação de teclas **Ctrl+G**.

Se o ficheiro `.bash_profile` for alterado, não é necessário sair do sistema e voltar a entrar para que as alterações tenham efeito imediatamente. Para isso, deve-se executar o comando `source`, seguido do nome do ficheiro especial de configuração:

```
source .bash_profile
```

Além do ficheiro `.bash_profile`, é possível haver outros dois ficheiros especiais de configuração que são executados sempre que uma nova sessão de trabalho *Linux* tenha início. São eles o `.bash_login` e o `.profile`. Repare-se que os nomes destes três ficheiros começam por um ponto ("**.**"), tornando-os "ocultos" para certos comandos do sistema operativo. Se estes ficheiros existirem em simultâneo no sistema, então a sua ordem de execução é a que se segue: em primeiro lugar é executado o `.bash_profile`, seguido do `.bash_login`, e por último é executado o `.profile`.

Para exemplificar um possível exemplo de alteração do ficheiro de configuração `.bash_profile`, vamos acrescentar-lhe um comando *Linux* que apresenta uma mensagem de boas-vindas sempre que uma nova sessão de trabalho tenha início. Para isso, é necessário utilizar um comando que envie texto para a saída padrão de dados. Esse comando é o `echo`, que simplesmente envia o texto que se lhe segue para o ficheiro virtual `stdout`:

```
echo "Teste de utilização do comando echo"
```

Se o comando `echo` for utilizado para enviar um texto constituído por um comando *Linux* normal, e a sua saída de dados for redireccionada para o fim do ficheiro de configuração `.bash_profile`, temos solucionado o problema proposto.

```
echo "echo 'Seja Bem-Vindo ao Linux!'" >> .bash_profile
```

Repare-se que o comando `echo` envia para o fim do ficheiro `.bash_profile` o texto correspondente a outro comando `echo`. Este é o comando que será executado quando o ficheiro `.bash_profile` for executado.

5.1.2 O Ficheiro .bashrc

Outro ficheiro especial de configuração leva o nome de `.bashrc`, e é executado sempre que se uma nova *"shell"* é executada, isto é, sempre que o utilizador decida trabalhar com outro interpretador de comandos. Esta situação é possível porque existem várias *"shells"*,

e qualquer utilizador pode escolher, em qualquer momento, a *"shell"* de que mais gosta (desde que essa *"shell"* esteja disponível no sistema operativo). Por omissão, a *"shell"* executada pelo sistema operativo *Linux* é a `bash`, uma sigla para a expressão inglesa *"Bourne Again shell"*. Para executar uma nova *"shell"*, basta executar o comando que leva o seu nome (`bash`, no caso da *"shell"* homónima).

Sempre que uma nova sessão de trabalho é iniciada, uma nova *"shell"* tem também a sua execução iniciada. Portanto, também durante o início de uma nova sessão de trabalho é executado o ficheiro de configuração `.bashrc`.

5.1.3 O Ficheiro .bash_logout

Por fim, existe ainda um ficheiro especial de configuração que é executado sempre que a sessão de trabalho *Linux* termina. Esse ficheiro tem o nome de `.bash_logout`. Para ilustrar a sua utilidade, vamos acrescentar a esse ficheiro um comando *Linux* que envie uma mensagem de despedida para a saída padrão de dados. Tal como foi visto em relação ao ficheiro de configuração `.bash_profile`, o exercício proposto é resolvido facilmente através de um comando `echo` que envia um outro comando `echo` para o ficheiro virtual `stdout`, que, por sua vez, é redireccionado para o fim do ficheiro de configuração `.bash_logout`:

```
echo "echo 'Adeus! :-)'" >> .bash_logout
```

5.2 Redefinição de Comandos

O sistema operativo *Linux* disponibiliza um mecanismo de redefinição de comandos, também conhecido por abreviaturas (*"alias"*) ou macros, que permite ao utilizador criar os seus próprios comandos e alterar a funcionalidade dos comandos já existentes. O seu formato genérico é o seguinte:

```
alias nome_comando="sequência de comandos"
```

Tipicamente, o utilizador deve guardar todas as suas redefinições de comandos no ficheiro especial de configuração `.bash_profile`, para que essas abreviaturas possam estar sempre disponíveis para as futuras sessões de trabalho desse utilizador.

Por exemplo, para redefinir o comando "`ls -l`" como a abreviatura "`l`", deve-se executar o seguinte comando:

```
alias l="ls -l"
```

Para obter uma lista de todas as redefinições de comandos actualmente em vigor, deve-se executar o comando `alias`, sem opções.

Em seguida, analisaremos mais alguns exemplos de redefinições de comandos. Por exemplo, outro comando muito utilizado durante as sessões de trabalho em *Linux* é `ls -la`. Pode-se definir uma abreviatura menor para facilitar a utilização deste comando:

```
alias la="ls -la"
```

Outras duas abreviaturas úteis, em especial para os utilizadores mais habituados com o sistema operativo MS-DOS, são as que envolvem os comandos `mkdir` e `rmdir`. Em MS-DOS, os comandos equivalentes têm nomes mais curtos: `md` e `rd`. Assim, duas novas abreviaturas podem ser criadas:

```
alias md=mkdir
alias rd=rmdir
```

5.3 Variáveis do Interpretador de Comandos

Uma das propriedades mais importantes dos interpretadores de comandos são as variáveis. Com a ajuda das variáveis, pode-se controlar muitos dos comportamentos da *"shell"* e dos programas executados a partir dela. Para consultar a listagem das variáveis já definidas, basta usar o comando `set`. Como é muito provável que essa listagem seja muito longa, é aconselhável lê-la através do comando `more`:

```
set | more
```

As variáveis da *"shell"* podem ser divididas em dois grupos: as variáveis internas, que apenas são conhecidas pela própria *"shell"*, e as variáveis do ambiente de trabalho, que são herdadas automaticamente por todos os programas executados a partir da *"shell"*.

Para criar uma variável interna, ou atribuir um novo valor a uma variável já existente, usa-se o operador "=", tal como se segue:

```
variável=valor
```

Para utilizar o valor de uma variável em conjunto com qualquer comando da *"shell"*, basta escrever o nome da variável, precedido do caracter "**$**". Quando o interpretador de comandos encontra um caracter "**$**" seguido do nome de uma variável, substitui imediatamente essa variável pelo seu respectivo valor. Por exemplo, para ver o valor de uma variável, pode-se usar um comando como o que se segue:

```
echo $variável
```

Por exemplo, vamos declarar uma nova variável, chamada NOVA_VAR, e atribuir-lhe um valor inicial:

```
NOVA_VAR="Linux em Portugal"
```

Para ver o valor dessa variável, vamos usá-la com o comando `echo`:

```
echo "A valor da variável é: " $NOVA_VAR
```

O resultado da execução do comando anterior é:

```
A valor da variável é: Linux em Portugal
```

Como já foi dito mais em cima, a utilização das variáveis revela-se realmente importante na configuração da própria *"shell"* e do ambiente de trabalho. Existem inúmeras variáveis que permitem o controlo de diversos aspectos de uma sessão de trabalho no sistema operativo *Linux*. Por exemplo, a variável interna MAIL é utilizada para guardar o nome do ficheiro onde é guardada a caixa de correio electrónico do utilizador. Se o utilizador quiser alterar o nome ou a localização desse ficheiro, apenas terá que atribuir o novo nome do ficheiro a esta variável. Por outro lado, também é possível ao utilizador controlar a periodicidade com que o sistema verifica se há novas mensagens de correio electrónico, através da variável interna MAILCHECK.

Uma variável interna muito útil é PS1, que permite configurar o aviso de comando (*"prompt"*) do interpretador de comandos. Para alterar o aviso de comando, é necessário atribuir à variável PS1 um valor constituído por alguns símbolos com significado especial para a *"shell"*. Esses símbolos podem ser resumidos na tabela que se segue:

Símbolo	Significado
\d	Data actual do sistema.
\h	Nome do computador ao qual o utilizador está ligado.
\t	Hora actual do sistema.
\u	Nome do utilizador.
\w	Directoria actual de trabalho.
\!	Número do comando actual, em relação à história dos comandos executados pelo utilizador.
\#	Número do comando actual, em relação à sessão de trabalho actual.

Por exemplo, se à variável interna PS1 for atribuído o valor "`[\# \u@\h \w]$`", o aviso de comando do utilizador terá o seguinte aspecto:

```
[13 reis.quarteu@elara ~]$
```

Através deste aviso de comando, é possível obter várias informações úteis:

- O próximo comando a ser introduzido será o **13**.º desde o início da sessão actual de trabalho;
- O nome do utilizador é **reis.quarteu**.
- O nome da máquina à qual esse utilizador está ligado é **elara**;
- A directoria actual de trabalho é a directoria pessoal (~) do utilizador.

Outra variável interna importante é PATH. Esta variável guarda uma lista de directorias, separadas pelo caracter ":", que são utilizadas pelo interpretador de comandos para procurar comandos, programas ou ficheiros. É devido à existência desta variável que não se torna necessário indicar o caminho completo para aceder aos comandos do sistema operativo, porque esses caminhos já se encontram, por omissão, definidos na variável interna PATH. Se o utilizador pretender acrescentar uma nova directoria a essa lista, basta actualizar essa variável da seguinte maneira:

```
PATH=$PATH:nova_directoria
```

Este comando obtém o valor actual da variável interna PATH, através da expressão $PATH; depois, acrescenta-lhe a expressão ":nova_directoria"; por fim, atribui o novo valor à mesma variável interna PATH.

5.4 Variáveis do Ambiente de Trabalho

As variáveis do ambiente de trabalho constituem um subconjunto das variáveis da *"shell"*, e que são conhecidas automaticamente por todo e qualquer programa executado a partir dessa *"shell"*. À partida, todas as variáveis criadas na *"shell"* são apenas variáveis internas da *"shell"*, não sendo conhecidas pelos programas e comandos executados pelo interpretador de comandos. Para que uma variável interna passe a ser também conhecida por esses programas, é necessário que ela seja exportada. Para isso, utiliza-se o comando `export`. Por exemplo, para exportar a variável NOVA_VAR utilizado num dos exemplos precedentes, deve-se executar o comando seguinte:

```
export NOVA_VAR
```

É possível criar, inicializar e exportar uma variável interna num único passo. Se porventura a variável NOVA_VAR ainda não existisse, poder-se-ia criá-la e exportá-la assim:

```
export NOVA_VAR="Linux em Portugal"
```

Para se obter uma lista com todas as variáveis de ambiente, basta executar o comando `export` sem quaisquer opções.

6. Gestão de Processos[29]

O sistema operativo *Linux* é um sistema operativo multi-utilizador e multi-tarefa. Desta forma, vários utilizadores podem estar a usar o sistema ao mesmo tempo e, por seu turno, cada utilizador pode executar várias tarefas em simultâneo. No entanto, até ao momento todos os exemplos que foram apresentados nesta sebenta executam sempre um comando de cada vez[30]. Diz-se que esses comandos são executados em primeiro plano (*"foreground"*).

Por outro lado, quando tarefas muito demoradas (e que, além disso, não interagem com o utilizador) estão a ser executadas, não há qualquer razão para que se tenha de esperar que a execução dessas tarefas termine. Em situações como estas, seria muito útil mandar executar essa tarefa e continuar a trabalhar noutras tarefas.

Por exemplo, imagine-se que se pretende criar uma listagem recursiva de todos os ficheiros de todas as directorias do sistema, e que essa listagem fosse gravada num ficheiro chamado "tree.txt", localizado na directoria pessoal do utilizador. O comando *Linux* necessário para executar esta tarefa é o seguinte:

```
ls -lR / > ~/tree.txt[31]
```

Dependendo da velocidade de cada computador e da dimensão do seu disco rígido, este comando pode demorar vários minutos a executar, ou até mesmo horas!

O utilizador pode terminar ou suspender a execução de uma tarefa que esteja a ser executada em primeiro plano. Para terminar, basta carregar a combinação de teclas **Ctrl+C**. Na maioria dos casos, isto basta para terminar a tarefa e para reactivar o interpretador de comandos[32].

Para suspender a execução de uma tarefa em primeiro plano, a combinação de teclas que deverá ser usada é **Ctrl+Z**. Desta forma, o processo referente a essa tarefa fica bloqueado (ou seja, a sua execução é momentaneamente interrompida, mas poderá ser reactivada a qualquer momento) e o controlo volta para o interpretador de comandos.

[29] Este capítulo destina-se apenas aos alunos do curso de Engenharia Informática.
[30] Excepto quando foram usados "tubos" (*"pipes"*). Nesses casos, o interpretador de comandos lança todos os programas ao mesmo tempo para que eles possam comunicar entre si.
[31] A opção `-R` do comando `ls` permite aceder, recursivamente, a todas as subdirectorias da directoria indicada; neste caso, a directoria raiz.
[32] Se esta solução não funcionar, experimente a combinação **Ctrl+**. Se esta também não resultar, então deverá usar o comando `kill`, o qual será discutido mais adiante.

Quando a execução de uma tarefa é suspensa, o sistema apresenta algumas informações referentes a essa tarefa:

```
[1]+ Stopped        ls -lR / > ~/tree.txt
```

A interpretação desta informação é a seguinte:

- "**1**" é o número da tarefa (*"job"*);
- "+" (ou "-") indica que, de todas as tarefas que temos actualmente (em execução ou suspensas), esta é aquela cujo estado mudou há menos (ou mais) tempo;
- *"Stopped"* é o estado actual da tarefa (neste caso, está suspensa);
- Por fim, o comando referente à tarefa cuja execução foi suspensa.

Uma tarefa cuja execução se encontre suspensa pode (e, muitas vezes, deve) voltar a ser executada. O sistema operativo *Linux* disponibiliza dois comandos para realizar essa tarefa, dependendo a escolha de qual deles usar em decidir se a tarefa deve ser executada em primeiro plano ou em segundo plano (*"background"*)[33]. Esses dois comandos são `fg` e `bg`, respectivamente.

Para voltar a executar em segundo plano uma tarefa cuja execução se encontre suspensa, deve-se usar o comando `bg`, seguido do número da tarefa pretendida. Se apenas existir uma tarefa, torna-se desnecessário indicar esse número. No nosso exemplo, para voltar a executar em segundo plano a tarefa entretanto suspensa, o comando que deveremos executar é o seguinte:

```
bg 1
```

A execução deste comando faz aparecer a seguinte informação:

```
[1]+ ls -lR / > ~/tree.txt &
```

Repare-se no caracter "**&**" que aparece no fim do comando: este é o símbolo que indica à *"shell"* que um comando deve ser executado em segundo plano. Assim, vê-se que

[33] Quando uma tarefa é executada em segundo plano, o interpretador de comandos fica imediatamente pronto para receber novos comandos. Assim, o utilizador tem a oportunidade de executar outras tarefas enquanto a tarefa em segundo plano é executada.

existe uma outra forma de executar um comando directamente em segundo plano a partir do interpretador de comandos: basta acrescentar o símbolo "**&**" no seu fim:

```
ls -lR / > ~/tree.txt &
```

Tal como está, o interpretador de comandos lança a tarefa e não fica à espera que ela termine. Em vez disso, o aviso de comando é imediatamente apresentado para que o utilizador possa continuar a trabalhar. Ao executar assim um comando, é exibida a seguinte informação:

```
[2] 874
```

O número entre parêntesis rectos representa o número da tarefa (admite-se aqui que o utilizador tem outra tarefa em execução). O 2.º número, 874, identifica o processo referente à tarefa em execução. Esse número é mais conhecido pela sigla **PID**: *"Process ID"* (Identificador de Processo).

Assim, é importante perceber, desde já, a diferença entre o número de tarefa e o número de processo:

- O número de processo é um identificador global, referente a todo o sistema operativo (*"system wide"*), do processo em execução.
- O número de tarefa é um identificador local de um processo, ao nível do utilizador (*"user wide"*).

É possível obter uma listagem de todas as tarefas do utilizador, através do comando `jobs`:

```
[1]-   Running      ls -lR / > ~/tree.txt &
[2]+   Running      ls -lR / > ~/tree.txt &
```

Neste exemplo, as duas tarefas estão em execução (estado *"Running"*), embora em segundo plano ("**&**").

Para se obter uma listagem das tarefas do utilizador, incluindo os seus números de processo, deve-se acrescentar ao comando `jobs` a opção `-l`:

```
[1]-    873   Running      ls -lR / > ~/tree.txt &
[2]+    874   Running      ls -lR / > ~/tree.txt &
```

Uma tarefa que esteja a ser executada em segundo plano pode ser trazida para primeiro plano através do comando `fg`, acompanhado pelo número da tarefa em causa. Por exemplo:

```
fg 2
```

Se o comando `fg` for executado sem um número de tarefa, então a tarefa que é trazida para primeiro plano é aquela que mudou de estado há menos tempo. Neste exemplo, seria a tarefa cujo número é 2 (veja-se o sinal "+" à sua direita).

Para terminar a execução de um processo que esteja em primeiro plano, basta utilizar a combinação de teclas **Ctrl+C**, como já foi discutido anteriormente. Para terminar a execução de um processo que esteja em segundo plano, existem duas alternativas:

1º) Trazer o processo para o primeiro plano, através do comando `fg`, e terminá-lo através da combinação de teclas **Ctrl+C**;

2º) Listas as tarefas, identificar o número do processo pretendido e terminá-lo através do comando `kill`.

O comando `kill` é utilizado para enviar sinais aos processos. Curiosamente, este nome foi-lhe atribuído porque os sinais enviados com maior frequência servem para "matar" (terminar) processos. Este comando pode enviar diversos tipos de sinais a um processo. Contudo, se o sinal não for especificado, é enviado o sinal que obriga o processo a terminar a sua execução. Assim, para terminar a execução do processo referente à tarefa 2, deve-se executar o seguinte comando:

```
kill -9 874
```

A opção –9 indica o sinal enviado ao processo (sinal KILL, que obriga o processo a "morrer"), e o número 874 é o PID do processo que se quer "matar".

É possível obter-se uma listagem com informação dos processos do utilizador através do comando `ps`. Conjugado com a opção `-l`, esse comando fornece também informação sobre o estado dos processos:

```
PID   TTY   STAT   TIME    COMMAND
162   p0    S      0:00    -bash
873   p0    R      9:54    ls
915   p0    R      0:00    ps
```

Nesta listagem, as colunas têm o seguinte significado:

PID: número de identificação do processo;

- TTY: nome do terminal onde o utilizador que lançou o processo estava a trabalhar;
- STAT: estado do processo:
  - S = *"Sleeping"* – parado à espera de ordens do utilizador;
  - R = *"Running"* – em execução;
  - T = *"Stopped"* – suspenso;
  - Z = *"Zombie"* – terminado;
  - D = *"Uninterruptible sleep"* – bloqueado numa operação de entrada e saída de dados.
- TIME: tempo de processador consumido pelo processo[34];
- COMMAND: nome do comando referente ao processo[35].

Para ver uma listagem com informação sobre todos os processos que estejam em execução na máquina em que se encontra a trabalhar, deve executar o comando `ps` conjugado com a opção `aux`:

```
ps aux
```

A figura **2** representa um resumo dos vários estados possíveis durante a execução de um processo, e como pode o utilizador alterar os estados desses processos. Um processo que seja iniciado a partir da *"shell"* pode ser executado em primeiro plano (*"foreground"*) ou em segundo plano (*"background"*), dependendo de o utilizador ter acrescentado o símbolo "**&**" ao fim do comando, ou não. Um processo que esteja em segundo plano pode passar para primeiro plano através do comando `fg`. Por sua vez, um processo que esteja em primeiro plano pode

[34] Se este valor for igual a zero, quer dizer que o processador apenas ocupou alguns milissegundos do tempo do processador.
[35] Repare-se que um dos comandos em execução é a `bash`, isto é, o próprio interpretador de comandos, que tinha a sua execução suspensa no momento em que a listagem foi gerada. A razão para isso é que o próprio comando `ps` (que também aparece na lista) estava em execução em primeiro plano, a preparar a lista dos processos, enquanto que a `bash` estava à espera que a execução do processo `ps` terminasse para que pudesse voltar a ser executada.

ser suspenso através da combinação de teclas **Ctrl+Z**. Além disso, um processo que esteja suspenso (*"stopped"*) pode passar para primeiro plano através do comando `fg`, ou pode passar para segundo plano através do comando `bg`. Por fim, um processo pode ser terminado através do comando `kill`, independentemente do estado em que se encontre (os processos que estejam em execução em primeiro plano poderão também ser terminados através da combinação de teclas **Ctrl+C**).

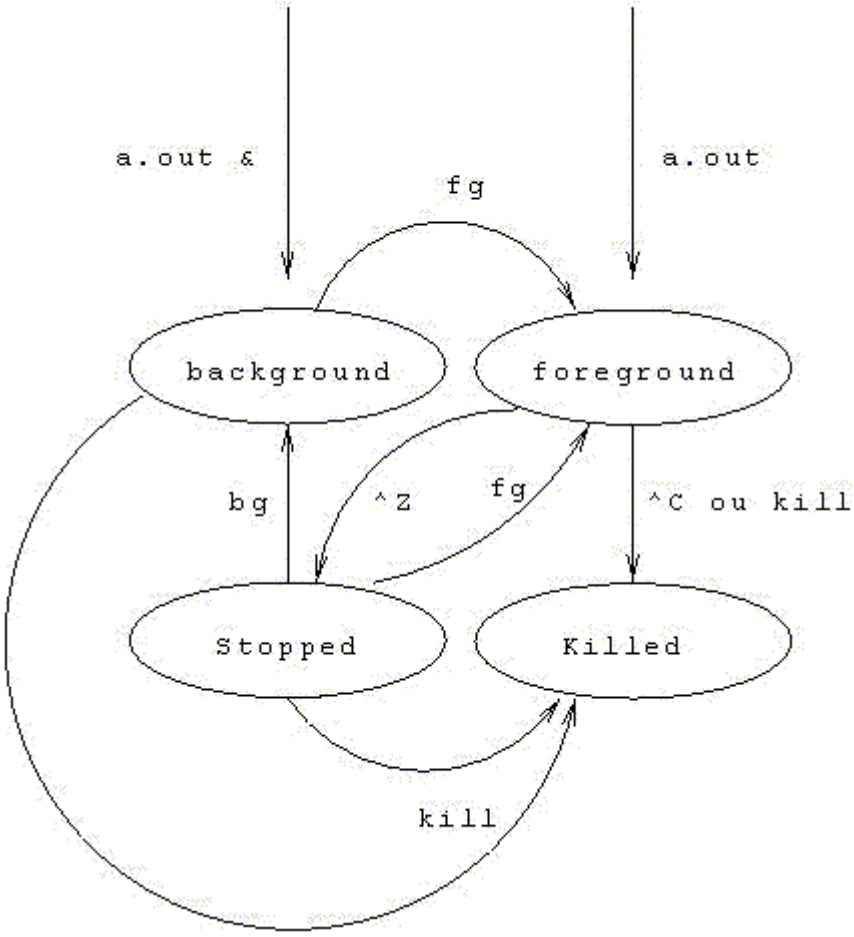


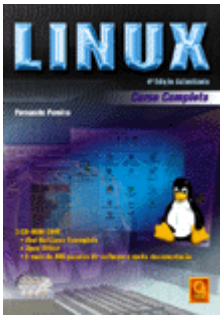
Figura 2: *Estados possíveis de processos em execução, e as transições entre esses estados.*

Por fim, uma pequena referência à execução simultânea de vários comandos. Existem três alternativas possíveis para lançar vários comandos de uma única vez:

- Em sequência: `comando1; comando2`
- Em sequência e em segundo plano: `(comando1; comando2) &`
- Em paralelo e em segundo plano: `(comando1 &); (comando2 &)`

Bibliografia

- Pereira, Fernando (2000): **Linux – Curso Completo**. FCA – Editora de Informática.



https://www.fca.pt/cgi-bin/fca_detalhado.cgi/?isbn=972-722-394-x