

Utilização de Comandos em Linux

Princípios da Computação
2008/2009

Outubro de 2008

Luis Lino Ferreira
Luis Nogueira
Berta Batista
Bertil Marques
Maria João Viamonte
Paulo Ferreira

Sugestões e participações de erros para:

llf@dei.isep.ipp.pt

AGRADECIMENTOS

Gostaríamos de agradecer aos colegas que permitiram a utilização do material pedagógico que serviu de base à elaboração deste texto, em particular Lino Oliveira, Orlando Sousa, Sandra Machado e Nuno Malheiro.

BIBLIOGRAFIA

“LINUX For Application Developers” de William A. Parrete, ISBN 007031697X, Editora Mcgraw-Hill, June 1, 1991

ÍNDICE

AGRADECIMENTOS	2
BIBLIOGRAFIA.....	2
ÍNDICE	3
1 A Shell do Linux	5
1.1 ALGUMAS CARACTERÍSTICAS DO Linux	5
1.2 CONFIGURAÇÃO DA BASH	6
1.3 OBTENÇÃO DE AJUDA	6
1.4 SINTAXE GERAL DOS COMANDOS Linux	7
1.5 ALTERAÇÃO DA PASSWORD.....	8
1.6 ABANDONAR A SHELL	9
2 FICHEIROS E DIRECTÓRIOS	9
2.1 DIRECTÓRIOS	9
2.1.1 DIRECTÓRIOS STANDARD	9
2.1.2 PATHNAMES	10
2.1.3 Comando pwd.....	10
2.1.4 DIRECTÓRIOS ESPECIAIS	10
2.1.5 MUDAR DE DIRECTÓRIO	11
2.1.6 CRIAR NOVOS DIRECTÓRIOS.....	11
2.1.7 ELIMINAR DIRECTÓRIOS.....	11
2.1.8 LISTAR NOME DE FICHEIROS.....	12
2.1.9 VER O CONTEÚDO DE UM FICHEIRO	12
2.1.10 VER O CONTEÚDO DE UM FICHEIRO PÁGINA A PÁGINA.....	13
2.1.11 VER A PARTE FINAL DUM FICHEIRO	13
2.1.12 VER A PARTE INICIAL DUM FICHEIRO	13
2.1.13 COPIAR UM FICHEIRO	14
2.1.14 MOVER OU RENOMEAR FICHEIRO	14
2.1.15 ELIMINAR FICHEIROS.....	14
3 PERMISSÕES	15
3.1 UTILIZADORES E GRUPOS.....	15
3.2 QUEM É VOCÊ E A QUE GRUPO PERTENCE	16
3.3 PERMISSÕES DE ACESSO	17
3.4 ALTERAR PERMISSÕES DE ACESSO	18
4 UTILIZAÇÃO DO INTERPRETADOR DE COMANDOS (SHELL)	19
4.1 INTERPRETAÇÃO E EXECUÇÃO DE COMANDOS	19
4.2 LOCALIZAÇÃO DOS COMANDOS.....	20
4.3 METACARACTERES DE EXPANSÃO	20
4.3.1 ASTERISCO (*)	20
4.3.2 PONTO DE INTERROGAÇÃO (?).....	21
4.3.3 PARÊNTESES RECTO ([]).....	21
4.3.4 CHAVETAS ({ })	21

4.3.5	TIL (~)	21
4.3.6	OS CARACTERES DE "DISFARCE" (\ , " , ' ')	21
4.4	METACARACTERES DE REDIRECÇÃO DE INPUT E OUTPUT	22
4.4.1	SINAL DE MENOR (<)	22
4.4.2	SINAL DE MAIOR (>)	22
4.4.3	DUPLO SINAL DE MAIOR (>>)	23
4.4.4	STDERR	23
4.5	PIPES E FILTROS	23
4.5.1	O COMANDO <i>tee</i>	24
4.6	EXEMPLOS DE APLICAÇÃO	26
5	MANIPULAÇÃO DE FICHEIROS DE TEXTO	26
5.1	CONTAGEM DE LINHAS NUM FICHEIRO DE TEXTO	26
5.2	SUBSTITUIR CARACTERES NUM FICHEIRO DE TEXTO	26
5.3	EXTRAIR COLUNAS NUM FICHEIRO DE TEXTO	26
5.4	"COLAR" FICHEIROS	27
6	COMANDOS AVANÇADOS (O USO DE EXPRESSÕES REGULARES)	28
6.1	INTRODUÇÃO ÀS EXPRESSÕES REGULARES	28
6.2	O COMANDO GREP	28
6.2.1	OPÇÕES DO COMANDO <i>grep</i>	29
6.3	O COMANDO <i>find</i>	29
7	Controlo de Processos em LINUX	34
7.1	Background vs Foreground	34
7.2	Executar Processos em Background; PIDs	34
7.3	Standard Input, Standard Output e Standard Error	34
7.4	O comando <i>Wait</i>	34
7.5	O comando <i>PS</i> - Process Status	34
7.6	O comando <i>Kill</i>	35
7.7	Prioridade de um Processo	36

1 A Shell do Linux

Uma *shell* oferece uma interface em formato texto com o *kernel* do sistema operativo. Permite realizar as operações mais habituais a um utilizador tal como: operações sobre ficheiros, gestão do sistema operativo, edição de texto, desenvolvimento de software, etc. Além disso esta tipo de interface apresenta como principal vantagem em relação a uma interface gráfica a possibilidade de correr programas feitos pelo utilizador utilizando uma linguagem de *script*. Em Linux existem várias *shells* à disposição do utilizador:

- **Bourne shell** (sh), a versão original presente em todos os sistemas Linux;
- **C shell** (csh), uma versão cujo nome deriva do facto de várias características de programação terem uma sintaxe semelhante à da linguagem de programação C. Tornou-se muito popular devido aos mecanismos de **alias** (permite criar nomes curtos para sequências de comando longas) e **history** (guarda os comandos executados e permite a sua re-execução);
- **Korn shell** (ksh) Tornou-se popular porque mantém a compatibilidade de sintaxe com a *Bourne shell* e ao mesmo tempo apresenta mecanismos de **alias** e **history** como a *C shell*;
- **Bourne Again shell** (bash) é uma *shell* que incorpora as características mais úteis da *Bourne shell* e da *C shell*. É uma das mais utilizadas comunidade de utilizadores de Linux, por isso será a única a ser utilizada na disciplina de Princípios da Computação.

Para abrir um terminal deve-se executar a aplicação *konsole*, se se estiver a utilizar o Linux em ambiente gráfico. Vai aparecer no ecrã uma janela onde se podem colocar comandos Linux ou lançar programas em execução.

1.1 ALGUMAS CARACTERÍSTICAS DO Linux

- **MULTI-UTILIZADOR:** apesar de existir apenas um computador, vários utilizadores podem estar ligados à máquina e o Linux faz parecer que cada um deles tem o seu próprio sistema operativo;
- **INTERACTIVIDADE:** espera a escrita de um comando, executa-o, apresenta o resultado e espera um novo comando;
- **MULTI-TAREFA:** cada utilizador tem a possibilidade de executar diversos utilitários (ou tarefas, como são normalmente designados) em simultâneo;
- **SEGURANÇA E PRIVACIDADE:** cada ficheiro num sistema Linux possui um conjunto de permissões associadas que impedem acessos indevidos. Cada utilizador tem também uma área de disco reservada que não interfere com mais ninguém;
- **INDEPENDÊNCIA DOS DISPOSITIVOS ENTRADA/SAÍDA:** uma característica importante do Linux reside no facto de que cada dispositivo periférico ligado ao computador é apenas mais um ficheiro para o sistema operativo Linux. Deste modo, todas as operações de escrita e leitura são executadas da mesma maneira, independentemente do dispositivo em causa;
- **COMUNICAÇÃO ENTRE PROCESSOS:** as aplicações podem ser desenvolvidas de maneira a ser possível a comunicação entre elas sem necessidade de intervenção do utilizador;
- **REDE:** o Linux possui todo o software necessário para instalar o computador numa rede. A partir desse momento, o Linux permite trabalhar com outros utilizadores de outros computadores, partilhando ficheiros e comunicando com eles;

- **COMANDOS/UTILITÁRIOS:** comandos são apenas programas utilitários. Esta flexibilidade permite-nos desenvolver os nossos próprios comandos e integrá-los nos sistemas operativo;
- **SHELL:** é uma característica do Linux que outros sistemas operativos têm vindo a “copiar”. De facto, a *shell* é apenas mais um utilitário que nos permite lançar os comandos a executar. Como utilitário que é, se não gostarmos da que nos é disponibilizada, podemos desenvolver a nossa própria *shell*. Disponibiliza também uma interface completamente programável.

1.2 CONFIGURAÇÃO DA BASH

A configuração da Bash vai para além, dos objectivos desta cadeira no entanto, dada a sua importância para quem utilizar o Linux frequentemente, transcreve-se abaixo um pequeno texto retirado do sítio <http://freeunix.dyndns.org:8088/site2/howto/Bash.shtml>, onde também poderá ser encontrada mais informação, por exemplo sobre como mudar o *prompt*.

In your home directory, 3 files have a special meaning to Bash, allowing you to set up your environment automatically when you log in and when you invoke another Bash shell, and allow you to execute commands when you log out.

These files may exist in your home directory, but that depends largely on the Linux distro you're using and how your sysadmin (if not you) has set up your account. If they're missing, Bash defaults to /etc/profile.

You can easily create these files yourself using your favorite texteditor. They are:

- .bash_profile : read and the commands in it executed by Bash every time you log in to the system
- .bashrc : read and executed by Bash every time you start a subshell
- .bash_logout : read and executed by Bash every time a login shell exits

Bash allows 2 synonyms for .bash_profile : .bash_login and .profile. These are derived from the C shell's file named .login and from the Bourne shell and Korn shell files named .profile. Only one of these files is read when you log in. If .bash_profile isn't there, Bash will look for .bash_login. If that is missing too, it will look for .profile.

.bash_profile is read and executed only when you start a login shell (that is, when you log in to the system). If you start a subshell (a new shell) by typing bash at the command prompt, it will read commands from .bashrc. This allows you to separate commands needed at login from those needed when invoking a subshell.

However, most people want to have the same commands run regardless of whether it is a login shell or a subshell. This can be done by using the source command from within .bash_profile to execute .bashrc. You would then simply place all the commands in .bashrc.

1.3 OBTENÇÃO DE AJUDA

Para além de material impresso (livros, guias de referência, revistas) ou acessível na Internet, poderemos encontrar ajuda sobre os comandos no próprio sistema Linux.

Todas as versões do Linux disponibilizam um comando **man** (abreviatura de manual) que nos permite obter informação detalhada sobre cada comando instalado no sistema. A ajuda obtém-se executando o comando da seguinte forma:

```
man comando
```

Este comando gera uma saída semelhante à existente nos manuais originais do sistema operativo.

Podemo-nos movimentar dentro do texto do comando `man` através das teclas:

Tabela 1 – Teclas do comando man (dependendo da configuração do terminal algumas destas teclas poderão não funcionar)

Enter	Mover o texto uma linha para baixo
Space	Mover o texto uma página para baixo
q	Sair
PageUp	Mover o texto meia página para cima
pageDown	Mover o texto meia página para baixo
End	Mover para o fim do texto
Home	Mover para o início do texto
/	Procura a primeira linha onde se encontra o texto escrito a seguir
n	Move-se para a próxima ocorrência do texto procurado
N	Move-se para a ocorrência anterior do texto procurado

Por vezes o mesmo comando poderá ser utilizado como um comando Linux ou, por exemplo, como uma função utilizada num programa em C. Nestes casos para se consultar outra versão do mesmo comando deve-se utilizar o comando:

```
man n comando
```

Em que *n* representa a versão alternativa da ajuda para esse comando.

Algumas distribuições do Linux permitem outras formas de obtenção de ajuda através do comando **help** que nos dá uma informação mais abreviada e que pode ser usado de duas maneiras:

```
help comando      ou      comando --help
```

Pode-se também obter os comandos relacionados com outro comando através da utilização do comando **apropos**. O exemplo seguinte refere-se à obtenção dos comandos relacionados com o comando **pwd**.

```
>apropos pwd
pwd          (1)  - print name of current/working directory
pwd          (n)  - Return the absolute path of the current working
directory
pwd [builtins] (1) - bash built-in commands, see bash(1)
pwdx         (1) - report current working directory of a process
```

1.4 SINTAXE GERAL DOS COMANDOS Linux

```
$ comando [ opção ... ] [ expressão ] [ ficheiro ... ]
```

\$ - *prompt*, indicativo de que estamos na *shell* (normalmente diferente para cada *shell*)
 [] - indicam que esta parte do comando é opcional
 ... - indicam que a parte em causa se pode repetir

opção	- parâmetros que condicionam a execução do comando
expressão	- dados necessários para a execução do comando
ficheiro	- se o comando opera com ficheiro(s) , estes aparecem sempre no fim do comando

Pode-se também executar vários comando seguidos utilizando como separador o ";" (ponto-e-vírgula):

```
$ comando1 ; comando2 ; ...
```

A *shell* executa-os todos, um de cada vez, como se eles tivessem sido introduzidos individualmente.

Alguns comandos Linux poderão necessitar de mais caracteres do que aqueles que podem ser apresentados no ecrã. A *shell* possibilita-nos lidar com este problema de duas maneiras:

- Quando chegarmos ao limite do ecrã, podemos continuar a escrever. Se o ecrã estiver correctamente configurado, os caracteres surgirão automaticamente no início da linha seguinte. Se não estiver correctamente configurado, o cursor ficará no limite do ecrã, e os caracteres serão continuamente apresentados na última posição da linha, à medida que os formos escrevendo. De qualquer maneira, a *shell* interpretará correctamente os caracteres introduzidos, sejam eles apresentados correctamente ou não;
- Outra maneira é finalizar a linha com um "\" (*backslash*) mesmo antes de pressionarmos a tecla "ENTER". O *backslash* dá indicações à *shell* de que o comando continua na linha seguinte. Exemplo:

```
bash> ls \<enter>
> -l <enter>
```

A alteração do *prompt* para ">" é uma indicação da *shell* de que está à espera da conclusão do comando.

1.5 ALTERAÇÃO DA PASSWORD

Um novo utilizador deve mudar a sua *password* quando fizer *login* a primeira vez. Existe um comando que permite fazer essa operação. É o comando **passwd**.

```
$ passwd
Changing password for lino
Old password:
New password
Re-enter password:
$
```

Sendo um dos pilares da segurança num sistema Linux, existem algumas regras que devem ser usadas na definição das *passwords*:

- Deve ter no mínimo 6 caracteres e existe um número máximo de caracteres que são considerados;
- Deve ser uma combinação de letras e números;
- Não pode ser o *username*, o seu inverso ou o *username* deslocado de um ou mais caracteres;
- Uma nova *password* deverá ser sempre diferente da anterior.

1.6 ABANDONAR A SHELL

Quando tivermos concluído o nosso trabalho no sistema, não devemos apenas abandonar o terminal onde estivemos. Isto pode constituir um problema grave uma vez que qualquer pessoa que o passe a utilizar, o fará com a nossa identidade no sistema e terá acesso aos nossos ficheiros e dados. Deveremos por isso informar a *shell* que pretendemos abandonar o sistema.

Como a *shell* é mais um dos programas Linux que lê os comandos como se o estivesse a fazer de um ficheiro, deveremos indicar o fim desse “ficheiro”. Isso faz-se com a introdução do “Control-D” que é o caracter de fim de ficheiro do sistema Linux.

Existe um comando que executa a mesma função que o “Control-D”. É o comando **exit**.

Em C *shell* existe o comando **logout**.

2 FICHEIROS E DIRECTÓRIOS

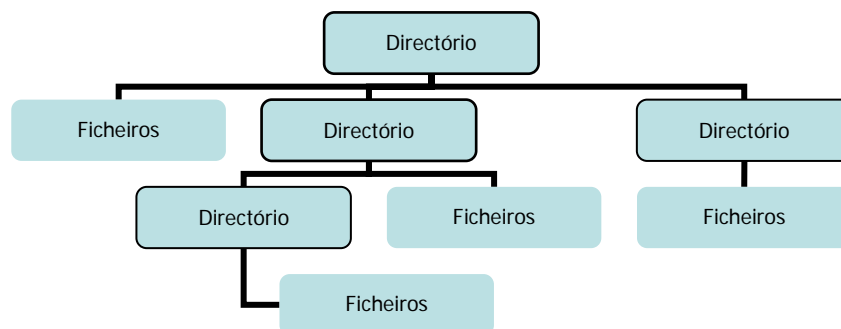
Os ficheiros e os directórios fazem parte do Sistema de Ficheiros.

O sistema de ficheiros é o local onde o sistema operativo e os utilizadores guardam e organizam os seus ficheiros.

O Linux impõe uma estrutura ao sistema de ficheiros que facilita o armazenamento e a procura posterior.

2.1 DIRECTÓRIOS

- É utilizado para guardar ficheiros relacionados num mesmo local;
- Pode conter ficheiros ou outros directórios;
- É tratado pelo Linux com um tipo especial de ficheiro;
- Não há limite para o número de níveis de directórios.
- O Sistema de Ficheiros é uma estrutura em árvore invertida.



2.1.1 DIRECTÓRIOS STANDARD

/	- directório raiz (<i>root</i>)
bin	- comandos principais do Linux
dev	- dispositivos (<i>devices</i>) ligados ao computador
etc.	- comandos e ficheiros usados pelo administrador do sistema
lib	- bibliotecas (conjunto de ficheiros relacionados contidos num único ficheiro) usadas por compiladores, processadores de texto e outros comandos Linux
lost+found	- usado pelo comando fsck (<i>file system check</i>). Este comando é usado apenas pelo administrador para verificar o sistema de ficheiros; se o fsck encontra algum ficheiro que

	pareça não estar ligado a nenhum directório, o comando liga-o ao <i>lost+found</i> , para que posteriormente o administrador decida o que fazer com ele
tmp	- usado por diversos comandos Linux para criação de ficheiros temporários; pode ser usado por qualquer utilizador; este directório é limpo regularmente
usr	- é a parte do sistema de ficheiros pertencente aos utilizadores; é a partir deste directório que se estendem os directórios de todos os utilizadores do sistema
home	- normalmente contém os directórios dos utilizadores
mnt	- utilizada para permitir o acesso a sistemas de ficheiros temporários (p.e. a uma <i>pen</i>)
Proc	- disponibiliza informação sobre os processo que se encontram a correr no sistema

Home Directory: directório ao qual temos acesso logo após o *login*. Em muitos sistemas, Linux, pode ser abreviado com o sinal "~", ou \$HOME

Current Working Directory (Directório corrente): directório onde "estamos" em cada momento. Quando fazemos *login*, o nosso *home directory* é simultaneamente o nosso *current working directory*.

2.1.2 PATHNAMES

- Referenciam um ficheiro ou directório no sistema de ficheiros
- Indicam o caminho a seguir através do sistema de ficheiro para encontrar o ficheiro ou directório

Tipos de *Pathnames*:

Full pathname	- caminho completo: nome do ficheiro ou directório em relação à raiz do sistema; começa sempre com "/" e apresenta cada um dos directórios separados por "/"; por cada ficheiro ou directório só existe um e um só <i>full pathname</i> ex: /users/home/i000001
Relative pathname	- caminho relativo: nome do ficheiro ou directório relativamente ao directório corrente de trabalho; nunca começa com "/" ex: cd sol/pratica
Simple pathname	- caminho simples: nome do ficheiro ou directório que está directamente abaixo, ou no interior, do directório corrente de trabalho ex: cd temp

2.1.3 Comando pwd

Podemos determinar o nosso directório de trabalho executando o comando **pwd** que nos apresenta o caminho completo desde a raiz (/) até ao nosso directório de trabalho. Por exemplo:

```
$ pwd
/users/2/lino
$
```

2.1.4 DIRECTÓRIOS ESPECIAIS

Estão presentes em cada um dos directórios:

"." - (ponto) abreviatura do directório corrente

".." - (ponto-ponto) abreviatura do directório imediatamente acima do corrente (directório pai)

2.1.5 MUDAR DE DIRECTÓRIO

O comando para mudar de directório é o **cd** (*change directory*) cuja sintaxe é a seguinte:

```
cd [ directório ]
```

O directório deve existir e temos de ter permissão para lá estar. Se não usarmos nenhum directório, isto é, se executarmos **cd** sem nenhum parâmetro, mudamos para a nossa *home directory*.

Para subirmos um nível na árvore do sistema de ficheiros podemos recorrer ao directório especial que representa o "pai" (..) fazendo:

```
cd ..
```

Se em qualquer altura, após várias mudanças de directório, não tivermos noção do local exacto onde nos encontramos no sistema de ficheiros, deveremos recorrer ao comando **pwd** (já referido).

2.1.6 CRIAR NOVOS DIRECTÓRIOS

É possível organizarmos os nossos ficheiros no nosso *home directory* da mesma maneira que o Linux o faz na *root directory*.

Tal é conseguido com o comando **mkdir** (*make directory*) cuja sintaxe é a seguinte:

```
mkdir directório ...
```

Os nomes a utilizar na criação de directórios seguem as mesmas regras que os nomes dos ficheiros: existe distinção entre maiúsculas e minúsculas. Algumas pessoas seguem a convenção de criar os directórios em maiúsculas o que permite uma mais fácil distinção dos ficheiros.

Se tentarmos criar um directório que já existe ou num sítio no qual não temos permissão para o fazer, **mkdir** apresentará uma mensagem de erro.

2.1.7 ELIMINAR DIRECTÓRIOS

Quando já não precisamos do directório podemos eliminá-lo com o comando **rmdir** cuja sintaxe é a seguinte:

```
rmdir directório ...
```

Para que um directório possa ser eliminado, duas condições devem ser respeitadas:

- o directório tem de estar vazio: é necessário apagar os ficheiros contidos dentro do directório a eliminar antes de proceder à operação
- o directório corrente não pode ser o directório que estamos a tentar eliminar; o que faz sentido: em que parte do sistema de ficheiros seríamos "colocados" se eliminássemos o nosso directório de trabalho corrente?

O **rmdir** não pede confirmação antes de proceder à eliminação. Como muitos outros comandos de Linux, o **rmdir** assume que sabemos o que estamos a fazer e deixa-nos fazer quase tudo o que queremos sem nos interromper com mensagens triviais!

2.1.8 LISTAR NOME DE FICHEIROS

O comando `ls` lista os ficheiros

```
ls [-aClqrstux] [nome ...]
```

a – (*all*) mostra ficheiros começados por “.”

l – (*long*) mostra em formato detalhado

h – mostra o tamanho dos ficheiros num formato mais perceptível para humanos (kB, MB, etc). Esta opção apenas funciona quando utilizado em conjunto com o `-l`, p.e. `ls -lh`.

t – ordena de acordo com a hora de modificação dos ficheiros

u – ordena de acordo com a hora de acesso dos ficheiros

R – lista recursivamente todas as directorias abaixo da directoria indicada em nome

Para vermos os ficheiros contidos nos directórios usamos o já referido comando `ls`. Quando usado sem parâmetros, permite ver os ficheiros e os directórios do directório corrente.

Eis mais alguns exemplos:

`ls dir_name` - lista os ficheiros contidos no directório **dir_name** e não o directório em si; se quisermos ver os ficheiros de um directório não precisamos de fazer `cd`

`ls -R` - modo recursivo: lista os directórios e respectivos conteúdos.

Exemplo: `ls -R / | more`

Existem ainda várias outras opções, para além das já referidas, que podem ser usadas com o comando `ls`. A opção “`-l`” dá-nos uma lista longa, isto é informação detalhada, de ficheiros e directórios. A seguir apresenta-se um exemplo do resultado possível do comando `ls -l` e a respectiva explicação:

<code>-r--r--r--</code>	<code>1</code>	<code>root</code>	<code>sys</code>	<code>5632</code>	<code>Apr 3 14:59</code>	<code>/etc/passwd</code>
↓	↓	↓	↓	↓	↓	↓
Permissões	N.º de links	Dono (owner)	Grupo	Tamanho	Data e hora da última alteração	Nome do ficheiro

2.1.9 VER O CONTEÚDO DE UM FICHEIRO

O comando `cat` (abreviatura de *concatenate and print*) permite visualizar o conteúdo de um ou mais ficheiros. A sintaxe do comando é a seguinte:

```
cat [-s] [-v [-t] [-e]] ficheiro ...
```

O comando `cat` “despeja” tudo para o ecrã, sem qualquer pausa ou separação entre os conteúdos dos diversos ficheiros.

A opção `-s` suprime quaisquer mensagens de erro ou aviso que possam ser geradas pelo comando.

O comando **cat** é usado para apresentar o conteúdo de ficheiros de texto (ASCII). Se, no entanto, houver caracteres de controlo invisíveis, eles podem ser vistos usando a opção **-v**, que faz com que esses caracteres sejam precedidos do acento circunflexo (^). Este modo “visual” pode ser usado com mais duas outras opções. A opção **-t** permite ver os caracteres *tab* com “Control-I” (^I). A opção **-e** podemos ver a posição de fim de linha (*end of line*) com o cifrão (\$).

Embora seja útil em muitas situações, não é o comando que é normalmente usado para ver o conteúdo de um ficheiro.

2.1.10 VER O CONTEÚDO DE UM FICHEIRO PÁGINA A PÁGINA

O comando **more** Permite visualizar o conteúdo de um ficheiro, um ecrã de cada vez. A forma mais simples do comando é

```
more ficheiro ...
```

Para avançar para a página seguinte pressionamos a barra de espaços e “Enter” para ver linha a linha.

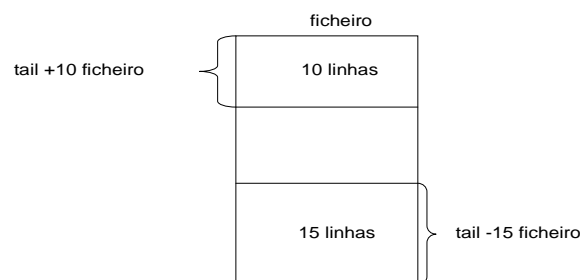
2.1.11 VER A PARTE FINAL DUM FICHEIRO

Outro comando útil para ver o conteúdo de um ficheiro é o **tail**. Por defeito, o **tail** apresenta as últimas 10 linhas do ficheiro. A sintaxe do comando é

```
tail [+|-número] [lbc] ficheiro
```

A seguir ao número indicamos **l** para linhas, **b** para blocos (1024 bytes) e **c** para caracteres (ou bytes). Se omitirmos a letra, o Linux assume que pretendemos linhas.

Por defeito, o **tail** executa o equivalente a **tail -10l**. O número fornecido como parâmetro indica o número de linhas que queremos ver. Se usarmos o sinal “-”, as linhas são contadas do fim para o princípio, para determinar o ponto a partir do qual começa a apresentar as linhas. Se usarmos o sinal “+”, as linhas são contadas do início para o fim.



2.1.12 VER A PARTE INICIAL DUM FICHEIRO

O comando **head** apresenta características semelhantes ao **tail**. A sintaxe é a seguinte:

```
head [-número] ficheiro ...
```

Por defeito, o comando **head** apresenta as 10 primeiras linhas. O número passado como parâmetro permite-nos indicar o número de linhas que queremos ver. Por exemplo, para vermos as primeiras 6 linhas do ficheiro:

```
$ head -6 ficheiro
```

2.1.13 COPIAR UM FICHEIRO

O Linux permite-nos efectuar cópias de ficheiros com o comando **cp** usando a sintaxe:

Cópia dum ficheiro para outro: `cp fich_orig fich_dest`

Cópia de ficheiro para directório: `cp fich_orig dir_dest`

Cópia de vários ficheiros para directório: `cp fich_orig1 fich_orig2 ... dir_dest`

Quando se copia um ficheiro para outro, o primeiro ficheiro é lido e copiado para o segundo ficheiro. Se o segundo ficheiro não existir, é criado. **Se existir, é substituído.** Como na maior parte dos comandos Linux, **cp** não pede confirmação antes de efectuar a cópia.

Quando se copia um ou mais ficheiros para um directório, cada ficheiro é lido individualmente e copiado para um ficheiro com o mesmo nome no directório indicado. O directório tem de existir. Se o ficheiro não existir nesse directório, é criado. **Se existir é substituído.**

2.1.14 MOVER OU RENOMEAR FICHEIRO

O Linux permite mover um ficheiro ou alterar o seu nome através do comando **mv**.

Mover ou alterar nome dum ficheiro para outro: `mv [-f] fich_orig fich_dest`

Mover ficheiro para directório: `mv [-f] fich_orig dir_dest`

Mover vários ficheiros para directório: `mv [-f] fich_orig1 fich_orig2 ... dir_dest`

Quando se “move” um ficheiro para outro, o primeiro ficheiro é lido e copiado para o segundo. O primeiro ficheiro é removido. Se o segundo ficheiro não existir, é criado. **Se existir, é substituído.** Como na maior parte dos comandos Linux, **mv** não pede confirmação.

Quando se “move” um ou mais ficheiros para um directório, cada ficheiro é lido individualmente e copiado para um ficheiro com o mesmo nome no directório indicado. Os ficheiros originais são removidos. O directório tem de existir. Se o ficheiro não existir nesse directório, é criado. **Se existir é substituído.**

Se se tentar mover um ficheiro para outro ficheiro ou directório, que nos pertence mas para o qual não temos permissão para alterar, o **mv** pede confirmação para completar a operação. A opção **-f** elimina estas mensagens de aviso e confirmação bem como quaisquer outras mensagens de erro ou aviso que possam ocorrer durante uma operação de “move”.

2.1.15 ELIMINAR FICHEIROS

O comando **rm** permite remover, ou eliminar um ficheiro. A sintaxe do comando é a seguinte:

```
rm [ -fir ] ficheiro ...
```

Cada ficheiro indicado no comando é removido do sistema de ficheiros. Mais uma vez, o Linux assume que sabemos o que estamos a fazer pelo que não efectua qualquer confirmação antes de proceder à execução do comando.

Algumas opções podem ser usadas para modificar a execução do comando **rm**:

- f Esta opção, força a execução do comando, isto é, neste caso, a eliminação do ficheiro. Nenhuma pergunta é feita, nem são apresentados quaisquer avisos ou mensagens de erro.
- i Esta opção é a opção de interactividade. Informa o **rm** para interagir com o utilizador durante a operação de eliminação. Antes do ficheiro ser efectivamente eliminado, o **rm** apresenta o nome do ficheiro seguido de um "?". Se se pretende eliminar, pressiona-se o "y" seguido do "Enter". Qualquer outra resposta "salta" a eliminação do ficheiro.
- r Esta opção permite ultrapassar a limitação do **rmdir** na eliminação de directórios não vazios. Com esta opção, o **rm** elimina qualquer directório indicado bem como os ficheiros neles contidos. O **rm** desce recursivamente ao longo da estrutura de directórios começando no directório indicado e elimina todos os ficheiros e directórios árvore acima, incluindo o directório inicial.

Devido ao facto do Linux ser um sistema multi-utilizador com utilizadores a criar e a eliminar ficheiros constantemente, é praticamente impossível recuperar um ficheiro eliminado. Logo que um ficheiro é eliminado, o sistema de ficheiros verifica que existe algum espaço livre que pode usar. No próximo pedido de espaço que alguém faça, o sistema de ficheiros provavelmente usará o espaço que acabou de libertar.

A única possibilidade de recuperação de um ficheiro eliminado é extrair uma cópia a partir de uma cópia de segurança efectuada recentemente.

Atenção ao comando **rm. Com apenas 5 teclas podemos colocar o nosso sistema de ficheiros Linux em apuros. O comando **rm *** remove tudo no directório corrente, e não nos será pedida qualquer confirmação.**

Pior do que isto, se estivermos *logged in* como administradores do sistema, nunca deveremos experimentar o comando **rm -r / . Se removermos tudo a partir da raiz, perderemos o nosso sistema de ficheiros por inteiro.**

Como se pode imaginar, o comando **rm** não remove efectivamente os dados. Remove uma entrada (um *link*) na tabela de ficheiros representativa do sistema de ficheiros (conhecida com FAT *File Allocation Table*). Quando o sistema de ficheiros verifica que não há mais *links* para o *i-node* em particular, liberta esse *i-node* e os blocos de disco a ele alocados para poder ser usado por outro ficheiro.

3 PERMISSÕES

O sistema de ficheiro do Linux possui alguns mecanismos básicos de permissões de acesso a ficheiros e directórios que permitem controlar quem tem acesso aos mesmos e que tipo de operações pode realizar.

3.1 UTILIZADORES E GRUPOS

Cada utilizador que tem acesso ao sistema Linux tem um *login id*, atribuído pelo administrador do sistema, que o identifica sem ambiguidade perante o sistema. Adicionalmente, cada utilizador pertence ao grupo de utilizadores relacionados que tem um determinado *group id*.

Um grupo é um conjunto de utilizadores que têm uma característica comum, por exemplo, pertencerem ao mesmo departamento. Um utilizador pode pertencer a mais do que um grupo.

Um dos motivos principais pelo qual os utilizadores estão relacionados através de um grupo é conceder-lhes a possibilidade de partilhar ficheiros. Deste modo, vários utilizadores poderão aceder e alterar aos mesmos ficheiros partilhando informação e tarefas de interesse comum.

Existe um ficheiro, normalmente no directório */etc*, onde estão registados os diversos utilizadores autorizados a aceder ao sistema bem como os grupos a que eles pertencem. Esse ficheiro é o */etc/passwd* e

é basicamente uma base de dados contendo um registo (linha) por cada utilizador. Cada linha é composta por registos separados por ":" que fornecem ao Linux informação importante de cada utilizador.

```
operator:x:11:0:operator:/var:/bin/sh
games:x:12:100:games:/usr/games:/bin/sh
nobody:x:65534:65534:Nobody:/:/bin/sh
```

Cada registo de */etc/passwd* apresenta os seguintes campos (indicados pela ordem em que aparecem):

- **Login id** – *login* que o utilizador usa para aceder ao sistema;
- **Password encriptada** – nenhuma *password* é guardada no sistema sem ser encriptada;
- **User id** – identificação numérica do utilizador;
- **Group id** – identificação numérica do grupo principal a que o utilizador pertence; informação acerca deste grupo, nomeadamente o seu nome, pode ser encontrada no ficheiro */etc/group*;
- **Comentário** – normalmente este campo é usado para o nome completo do utilizador;
- **Caminho completo para o *home directory* do utilizador** - este campo é usado pelo sistema durante o processo de *login* para determinar qual o local do sistema de ficheiros onde seremos "colocados";
- **Caminho completo do programa inicial** – isto identifica qual o programa a ser executado logo após o processo de *login*; normalmente é identificada aqui a *shell* usada pelo utilizador.

Outro ficheiro importante é o */etc/group*. Enquanto que no ficheiro */etc/passwd* é identificado o grupo inicial a que cada utilizador pertence, o */etc/group* especifica quais os grupos disponíveis.

Este ficheiro é também uma base de dados semelhante a */etc/passwd* e é composto pelos seguintes campos:

- **Group name** – da mesma maneira que cada utilizador tem um nome, cada grupo tem também um nome para mais fácil identificação;
- **Password encriptada** – o Linux permite *passwords* para grupos da mesma maneira que para utilizadores, de tal maneira que se queremos pertencer ao um grupo teremos que conhecer a respectiva password;
- **Group id** – identificação numérica do grupo e que é usada no ficheiro */etc/passwd*; para sabermos a que grupo o utilizador pertence, procuramos neste ficheiro a linha correspondente ao *group id* definido no registo do utilizador em */etc/passwd*;
- **Lista de login id's** – esta é a lista de utilizadores autorizados a pertencer a um determinado grupo.

Actualmente, é frequente em ambientes com mais de um sistema, existir um mecanismo de autenticação único. Assim consegue-se uma gestão centralizada pelo administrador do sistema. Exemplo desta abordagem é o nosso DEI, pelo que os ficheiros */etc/passwd* e */etc/group* não são os utilizados na autenticação existindo apenas para outras funcionalidades.

3.2 QUEM É VOCÊ E A QUE GRUPO PERTENCE

Podemos saber qual é o nosso *login*, *user id*, *group name* e *group id* sem ter necessidade de pesquisar o ficheiro */etc/passwd*. O comando **id** apresenta esta informação.

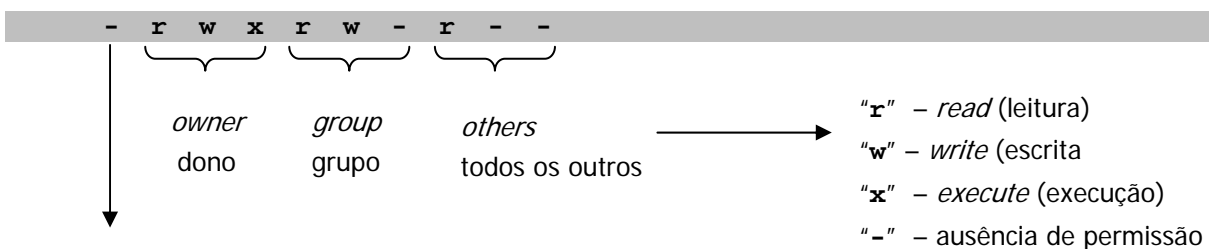
A sintaxe do comando `id` é a seguinte:

`id`

3.3 PERMISSÕES DE ACESSO

Sempre que um ficheiro ou directório é criado, o sistema operativo Linux atribui-lhe um conjunto de permissões de acesso. Estas permissões estão definidas por defeito, mas esse valor de defeito pode ser alterado pelo utilizador para os seus ficheiros. Também em qualquer altura o utilizador pode alterar as permissões dos seus ficheiros e directórios como se verá nas próximas secções.

As permissões são conjuntos de 10 caracteres divididos em 4 conjuntos que podem ser visualizadas no primeiro campo impresso no ecrã pelo comando `ls -l`.



tipo:

- `"-"` – ficheiro de dados
- `"d"` – directório
- `"c"` – dispositivo de caracteres
- `"b"` – dispositivos de blocos
- `"p"` – ficheiro *pipe*
- `"l"` – *link*

Podemos constatar que é possível atribuir diferentes permissões a diferentes tipos de utilizadores:

- *Owner* – o dono do ficheiro é normalmente a pessoa que o criou
- *Group* – o grupo do ficheiro é normalmente o grupo a que o utilizador pertencia no momento em que criou o ficheiro, e refere-se aos utilizadores que pertencem ao mesmo grupo que o dono
- *Others* – os outros são simplesmente todas as pessoas que não são o dono nem pertencem ao mesmo grupo que ele

Cada uma das 3 posições dos caracteres na secção das permissões tem um significado especial que corresponde a 3 diferentes tipos de permissões que cada ficheiro ou directório possui e tem um determinado significado:

- *Read* – a existência da permissão de leitura indica que é possível ler o conteúdo do ficheiro
- *Write* – a existência da permissão de escrita indica que é possível escrever ou alterar o conteúdo do ficheiro
- *Execute* – a existência da permissão de execução indica que é possível, pelo menos, fazer uma tentativa para executar ("correr") o ficheiro como um comando Linux ou um utilitário

No exemplo apresentado acima (`-rwxrwx-r--`), as permissões definidas são:

- *Owner* (`rwx`) – leitura (`r`), escrita (`w`) e execução (`x`)

- *Group* (*rw-*) – leitura (*r*) e escrita (*w*)
- *Others* (*r--*) – leitura (*r*)

Estas permissões são óbvias quando se referem a **ficheiros**: é necessário ter permissão de *read* ("*r*") para fazer o *cat* do conteúdo do ficheiro, é necessário ter permissão de *write* ("*w*") para editar e é necessário ter permissão de *execute* ("*x*") nos ficheiros criados executáveis ou contendo *scripts* Linux para que seja possível "executá-los".

Mas já não são tão óbvias quando se referem a **directórios**: *read* ("*r*") permite ver o conteúdo do directório, isto é, os ficheiros nele contidos, *write* ("*w*") permite copiar ficheiros para o directório ou apagá-los do directório, *execute* ("*x*") também designada permissão de pesquisa permite "ir para" o directório com o comando *cd* ou usar o directório em qualquer tipo de *pathname*.

3.4 ALTERAR PERMISSÕES DE ACESSO

Se as permissões atribuídas por defeito pelo sistema operativo não forem adequadas, podemos alterá-las como o comando **chmod** (*change mode*). Este comando permite alterar as permissões de cada ficheiro ou directório que nos pertença.

A sintaxe do comando é a seguinte:

```
chmod expressão_permissões ficheiro ...
```

A "expressão_permissões" permite definir as novas permissões que queremos atribuir ao(s) ficheiro(s) indicados no comando.

Esta expressão pode ser especificada através um número octal de 3 dígitos cujo significado é indicado de seguida:

Octal	Tipo de Permissão	Representação Linux	Binário
0	nenhuma	-	(000)
1	execute	x	(001)
2	write	w	(010)
3	write e execute	wx	(011)
4	read	r	(100)
5	read e execute	rx	(101)
6	read e write	rw	(110)
7	read, write e execute	rwX	(111)

Exemplo de utilização do comando **chmod**:

```
$ chmod 645 fich_temp
$ ls -l fich_temp
-rw-r--r-x  1 lino      profs    1123  Feb 17 19:10  fich_temp
$ chmod 440 fich_temp
$ ls -l fich_temp
-r--r-----  1 lino      profs    1123  Feb 17 19:10  fich_temp
```

Existe outro tipo de expressão que pode ser usado com o comando **chmod**. Esse tipo de expressão usa mnemónicas para especificar permissões.

Uma série de códigos mnemónicos podem ser usados em vez de números em octal e são apresentados nos quadros seguintes:

Classes de utilizadores

- u *User* (dono)
- g *Group* (grupo)
- o *Others* (outros)
- a *All* (todos)

Operações de permissões

- + adiciona
- retira
- = estabelece

Valores das permissões

- r *read*
- w *write*
- x *execute*

A utilização destes códigos torna mais fácil e intuitiva a atribuição das permissões, como se pode verificar pelo exemplo seguinte:

```
$ chmod u=rw ficheiro1
$ chmod go-rwx ficheiro2
```

Nestes exemplos são executadas as seguintes operações:

- **u=rw** – estabelece as permissões de leitura (*read "r"*) e escrita (*write "w"*) para o dono do ficheiro *ficheiro1*, independentemente das permissões que esse ficheiro tenha antes
- **go-rwx** – retira as permissões de leitura (*read "r"*), escrita (*write "w"*) e execução (*execute "x"*) para os utilizadores do grupo (*group "g"*) e para todos os outros (*others "o"*) ao ficheiro *ficheiro2*; as permissões do dono do ficheiro permanecem inalteradas.

Este método alternativo de estabelecimento de permissões tem ainda outra vantagem. Permite-nos conjugar diferentes expressões num único comando, separando-as com vírgulas, como a seguir se exemplifica:

```
$ chmod g+w, o-r ficheiro1
$ chmod u+x, g-w, o=r ficheiro2
```

4 UTILIZAÇÃO DO INTERPRETADOR DE COMANDOS (SHELL)

4.1 INTERPRETAÇÃO E EXECUÇÃO DE COMANDOS

Após a introdução de um comando a *shell* divide o comando em palavras (considerando como separador entre palavras o carácter "espaço"). De seguida define o que é comando, opções e nomes de ficheiros. Isto designa-se por executar o *parsing* do comando. Também procura metacaracteres, ou seja, caracteres com significado especial, como por exemplo o asterisco (*).

Metacaracter: carácter com um significado especial para a shell. Quando a shell encontra um metacaracter vai tomar as acções que conhece para esse carácter. Podem ser acções de expansão (substituição por um

conjunto de caracteres), de redireccionamento, etc. Há também metacaracteres que permitem que não se interpretem os metacaracteres (metacaracteres de "disfarce").

O próximo passo é o de carregar para memória o comando, passar-lhe todas as opções e nomes de ficheiros e esperar que o comando termine. Enquanto o comando é processado pelo sistema operativo a *shell* fica bloqueada.



4.2 LOCALIZAÇÃO DOS COMANDOS

Um comando é um ficheiro no sistema de ficheiros e como tal, para ser executado, é necessário que este seja localizado previamente. Contudo esta localização não é uma preocupação do sistema operativo Linux mas sim da *shell*. A *shell* é responsável por encontrar os comandos pelo que é necessário um mecanismo que possibilite que ela obtenha tal informação.

Para encontrar comandos a *shell* recorre a uma lista de directórios onde deve ir procurar os comandos. A essa lista chama-se *path* e está armazenada numa variável de ambiente com o mesmo nome. A pesquisa de um comando é efectuada sequencialmente na *path*; caso não seja encontrado a *shell* reporta que não encontrou o comando ("*Command not found.*"), o que não significa necessariamente que este não existe, mas apenas que não se encontra em nenhum directório da lista.

No caso de se indicar o *pathname* completo, por exemplo */bin/ls -l*, a *shell* não vai recorrer à variável *path*.

4.3 METACARACTERES DE EXPANSÃO

A maioria dos comandos que operam com ficheiros pode fazê-lo com vários ficheiros ao mesmo tempo. Se os ficheiros estão relacionados ou têm nomes semelhantes (Ex.: *capitulo1*, *capitulo2*, etc.) pode tirar-se partido de tais padrões de semelhança recorrendo a metacaracteres de expansão.

O comportamento da *shell* quando encontra um metacaracter é o seguinte:

1. Retira a palavra com o metacaracter.
2. Procura ficheiros no sistema de ficheiros que correspondem ao padrão.
3. Constrói um lista com esses ficheiros.
4. Introduce a lista no lugar da palavra que continha o metacaracter.

De seguida apresentam-se os metacaracteres mais comuns:

4.3.1 ASTERISCO (*)

Este caracter especial de expansão pode ser usado em qualquer posição de um nome ou *pathname* e representa **zero** ou **mais ocorrências** de qualquer caracater.

Ex.: «*lis*ta*» poderá equivaler a qualquer uma das seguintes: «*lista*», «*lis6ta*», «*lisrararta*», etc., desde que existam.

4.3.2 PONTO DE INTERROGAÇÃO (?)

Também pode ser usado em qualquer posição de um nome de um ficheiro ou *pathname*, mas apenas representa **uma ocorrência** de um qualquer carácter.

Ex.: «lista?» poderá equivaler a qualquer uma das seguintes: «lista1», «lista2», «listaA», «listaB», etc., desde que existam.

4.3.3 PARÊNTESES RECTO ([])

Também podem ser usados em qualquer posição de um nome de um ficheiro ou *pathname*, mas apenas representa **uma ocorrência de um carácter que conste entre os parêntesis rectos**.

Ex.: «lista[123]» poderá equivaler a qualquer uma das seguintes: «lista1», «lista2», «lista3», mas apenas a estas e desde que existam.

Este conceito também pode ser utilizado para representar intervalos de letras ou números, recorrendo a um hífen para representar um intervalo. Por exemplo, todas as letras minúsculas podem ser representadas por **[a-z]**, como analogamente todos os números, ou apenas parte de um intervalo de letras ou números, por exemplo **[5-15]**.

4.3.4 CHAVETAS ({ })

Este modo é semelhante ao anterior, contudo permite que sejam definidos não apenas um conjunto de possibilidades para um carácter mas um conjunto de possibilidades para vários conjuntos de caracteres, em que cada conjunto está separado dos restantes por vírgulas.

Ex.: «lis{ta,tas,tar}» poderá equivaler a qualquer dos seguintes, caso existam : «lista», «listas», «listar».

4.3.5 TIL (~)

Este carácter especial de expansão representa o *pathname* completo do *homedirectory* do utilizador.

4.3.6 OS CARACTERES DE "DISFARCE" (\ , " ", ' ')

Se por qualquer razão não se pretender que a *shell* interprete e substitua os metacaracteres com significado especial deve-se recorrer a caracteres de "disfarce". Assim:

\ - retira o significado ao carácter seguinte
 " " - retira o significado a todos os caracteres entre as aspas, excepto aos caracteres \$, ' e \
 ' ' - retira o significado a todos os caracteres entre as plicas

É preciso notar que se quisermos usar os caracteres de disfarce também temos de os disfarçar:

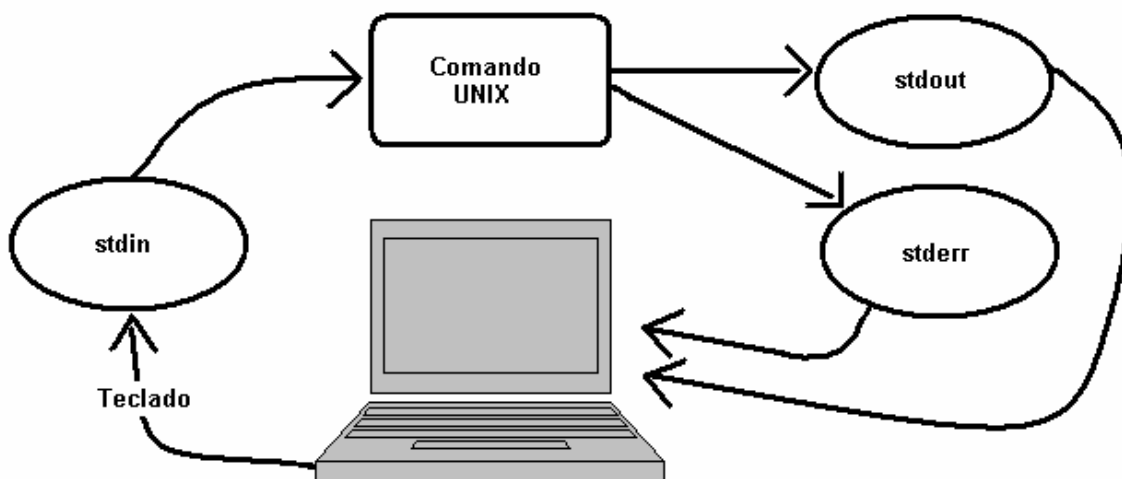
```
$ echo hi \ there
$ hi \ there
$ echo "hi \ there"
$ hi \ there
```

4.4 METACARACTERES DE REDIRECÇÃO DE INPUT E OUTPUT

Em Linux qualquer periférico é representado por um ficheiro no sistema de ficheiros, então desde que se possa trabalhar com ficheiros poder-se-á trabalhar com qualquer dispositivo suportado pelo Linux.

Todos os comandos do sistema Linux foram desenvolvidos para trabalhar com três ficheiros standard especiais, que são:

- Standard input (*stdin*): Se o comando necessita de informação durante a execução, deverá recorrer a este ficheiro para obter essa informação;
- Standard output (*stdout*): Se um comando gera informação deverá dirigi-la para este ficheiro;
- Standard error (*stderr*): Se um comando gera informação relativa a erros de execução deverá dirigi-la para este ficheiro.



Muitos dos comandos que usam um ficheiro de entrada de informação usam, por omissão, o *stdin* (teclado). Como quase todos os comandos que produzem informação fazem-no, por omissão, para o *stdout* (ecrã / terminal). O mesmo se passa com mensagens de erro, apesar de estas surgirem no terminal estão, por omissão, dirigidas para o *stderr* (que, como o *stdout*, está direccionado para o ecrã / terminal).

É possível redireccionar qualquer um destes ficheiros recorrendo a caracteres especiais, ou seja, metacaracteres de redireccionamento.

4.4.1 SINAL DE MENOR (<)

Permite alterar a entrada de informação num comando do standard *input* (teclado) para um ficheiro qualquer. Deverá ser colocado o sinal de menor depois do comando e de seguida o nome do ficheiro que contém a informação a fornecer ao comando.

```
Ex.: write mary < invitation
```

4.4.2 SINAL DE MAIOR (>)

Permite alterar a saída de informação de comando do standard *output* (ecrã) para um ficheiro. Deverá ser colocado o sinal de maior depois do comando e de seguida o nome do ficheiro onde queremos que seja armazenada a informação gerada pelo comando.

Nota: Caso o ficheiro exista o seu conteúdo é substituído com a informação gerada pelo comando. Se o ficheiro não existir este é criado automaticamente.

```
Ex.: date > dfile
```

4.4.3 DUPLO SINAL DE MAIOR (>>)

O funcionamento é praticamente o mesmo do anterior (sinal de maior), com a diferença deste não eliminar o conteúdo do ficheiro quando este já existe. Neste caso, adiciona a informação gerada pelo comando ao final do ficheiro (*append*), mantendo a informação que já existia no ficheiro.

Ex.: `who >> dfile`

4.4.4 STDERR

A designação do *standard error* difere em função da *shell* utilizada:

- Bash e Kourne shell: **2>** (**2>>** para modo append).
- C shell: **>&** (**>>&** para modo append).

Ex.: `ls -z 2> errors`

4.5 PIPES E FILTROS

Se se pretender usar a informação gerada por um comando (*stdout*) como entrada de informação (*stdin*) num outro comando, pode-se usar um ficheiro temporário. O exemplo seguinte mostra uma forma de se obter a lista ordenada dos utilizadores ligados no sistema, recorrendo a um ficheiro temporário.

```
aspinto@polux:~> who
ei8020036 pts/1      Nov  4 15:46 (192.168.3.77)
ei8020018 pts/0      Nov  4 15:46 (192.168.3.56)
ei8020079 pts/2      Nov  4 16:03 (192.168.3.65)
ei8010099 pts/3      Nov  4 15:47 (192.168.3.60)
ei8000019 pts/4      Nov  4 15:47 (192.168.3.66)
ei8020047 pts/5      Nov  4 15:48 (192.168.3.57)
ei8010031 pts/6      Nov  4 15:50 (192.168.3.54)
ei8010030 pts/7      Nov  4 15:52 (192.168.3.60)
ei8020064 pts/8      Nov  4 15:55 (192.168.3.102)
ei8020034 pts/9      Nov  4 16:00 (192.168.3.69)
aspinto  pts/10     Nov  4 16:04 (192.168.3.58)
aspinto@polux:~> who > lista.txt
aspinto@polux:~> sort < lista.txt
aspinto  pts/10     Nov  4 16:04 (192.168.3.58)
ei8000019 pts/4      Nov  4 15:47 (192.168.3.66)
ei8010030 pts/7      Nov  4 15:52 (192.168.3.60)
ei8010031 pts/6      Nov  4 15:50 (192.168.3.54)
ei8010099 pts/3      Nov  4 15:47 (192.168.3.60)
ei8020018 pts/0      Nov  4 15:46 (192.168.3.56)
ei8020034 pts/9      Nov  4 16:00 (192.168.3.69)
ei8020036 pts/1      Nov  4 15:46 (192.168.3.77)
```

```

ei8020047 pts/5      Nov  4 15:48 (192.168.3.57)
ei8020064 pts/8      Nov  4 15:55 (192.168.3.102)
ei8020079 pts/2      Nov  4 16:03 (192.168.3.65)
aspinto@polux:~>

```

Um **pipe** permite que esta operação seja feita sem recorrer a um ficheiro auxiliar, redireccionando automaticamente a saída de um comando para a entrada de outro.

A representação de um **pipe** é uma barra vertical (|).

O resultado do exemplo seguinte é o mesmo do anterior só que sem se recorrer ao ficheiro temporário.

```

aspinto@polux:~> who | sort
aspinto  pts/10    Nov  4 16:04 (192.168.3.58)
ei8000019 pts/4      Nov  4 15:47 (192.168.3.66)
ei8010030 pts/7      Nov  4 15:52 (192.168.3.60)
ei8010031 pts/6      Nov  4 15:50 (192.168.3.54)
ei8010099 pts/3      Nov  4 15:47 (192.168.3.60)
ei8020018 pts/0      Nov  4 15:46 (192.168.3.56)
ei8020034 pts/9      Nov  4 16:00 (192.168.3.69)
ei8020036 pts/1      Nov  4 15:46 (192.168.3.77)
ei8020047 pts/5      Nov  4 15:48 (192.168.3.57)
aspinto@polux:~>

```

Podem-se encadear mais do que dois comandos com recurso a **pipes**, criando-se um *pipeline*. Quando não existem comandos que façam exactamente aquilo que se pretende, pode-se criar um *pipeline*, tratando a informação sequencialmente de modo a que o resultado final seja o pretendido.

Contudo há que ter em atenção que certos comando apenas podem surgir no **início** de um *pipeline*, pela razão de que apenas geram informação e não necessitam de informação do utilizador (ex.: **ls**, **who**, **date**, **pwd**, **banner**, **file**, etc.).

Também há comandos que apenas podem surgir no **fim** de um *pipeline*, aqueles que apenas recebem informação mas que não geram informação para o ecrã (ex.: **mail**, **write**, **lp**, etc.).

Aos comandos que tanto geram informação para o ecrã como solicitam informação ao utilizador, chamam-se **filtros**. Os filtros podem surgir em qualquer parte de um *pipeline*, porque recebem informação do utilizador, processam essa informação e geram uma saída de informação para o ecrã.

4.5.1 O COMANDO **tee**

O comando **tee**, é um comando especial que foi desenvolvido para ser utilizado em **pipelines**. O que o comando faz é gerar um ou mais ficheiros com a informação que recebe e reenviar essa mesma informação para o comando seguinte num *pipeline*.

```
$ tee [OPTION]... [FILE]...
```

No exemplo seguinte, utilizando um **pipeline**, serão gerados dois ficheiros, um com o resultado do comando **who** e outro o resultado da ordenação do resultado do comando **who**.


```
aspinto@polux:~> who | tee listaWho | sort > listaWhoOrdenada
```

```
aspinto@polux:~> cat listaWho
```

```
ei8020036 pts/1      Nov  4 15:46 (192.168.3.77)
ei8020018 pts/0      Nov  4 15:46 (192.168.3.56)
ei8020079 pts/2      Nov  4 16:03 (192.168.3.65)
ei8010099 pts/3      Nov  4 15:47 (192.168.3.60)
ei8000019 pts/4      Nov  4 15:47 (192.168.3.66)
ei8020047 pts/5      Nov  4 15:48 (192.168.3.57)
ei8010031 pts/6      Nov  4 15:50 (192.168.3.54)
ei8010030 pts/7      Nov  4 15:52 (192.168.3.60)
ei8020100 pts/8      Nov  4 16:13 (192.168.3.107)
ei8020034 pts/9      Nov  4 16:00 (192.168.3.69)
aspinto   pts/10     Nov  4 16:04 (192.168.3.58)
ei8020064 pts/11     Nov  4 16:14 (192.168.3.102)
ei8020056 pts/12     Nov  4 16:09 (192.168.3.85)
ei8010115 pts/13     Nov  4 16:15 (192.168.3.72)
ei8020099 pts/14     Nov  4 16:18 (192.168.3.90)
ei8020008 pts/15     Nov  4 16:24 (192.168.3.61)
ei8010115 pts/16     Nov  4 16:28 (192.168.3.72)
```

```
aspinto@polux:~> cat listaWhoOrdenada
```

```
aspinto   pts/10     Nov  4 16:04 (192.168.3.58)
ei8000019 pts/4      Nov  4 15:47 (192.168.3.66)
ei8010030 pts/7      Nov  4 15:52 (192.168.3.60)
ei8010031 pts/6      Nov  4 15:50 (192.168.3.54)
ei8010099 pts/3      Nov  4 15:47 (192.168.3.60)
ei8010115 pts/13     Nov  4 16:15 (192.168.3.72)
ei8010115 pts/16     Nov  4 16:28 (192.168.3.72)
ei8020008 pts/15     Nov  4 16:24 (192.168.3.61)
ei8020018 pts/0      Nov  4 15:46 (192.168.3.56)
ei8020034 pts/9      Nov  4 16:00 (192.168.3.69)
ei8020036 pts/1      Nov  4 15:46 (192.168.3.77)
ei8020047 pts/5      Nov  4 15:48 (192.168.3.57)
ei8020056 pts/12     Nov  4 16:09 (192.168.3.85)
ei8020064 pts/11     Nov  4 16:14 (192.168.3.102)
ei8020079 pts/2      Nov  4 16:03 (192.168.3.65)
ei8020099 pts/14     Nov  4 16:18 (192.168.3.90)
ei8020100 pts/8      Nov  4 16:13 (192.168.3.107)
```

```
aspinto@polux:~>
```

4.6 EXEMPLOS DE APLICAÇÃO

Verifique a equivalência entre os *pipes* e os conjuntos de comandos seguintes:

```
$ ls -c | tee f1
```

```
$ ls -c > f1
```

```
$ cat f1
```

```
$ ls -c | tee f1 f2 > f3
```

```
$ ls -c > f1
```

```
$ cp f1 f2
```

```
$ cat f1
```

```
$ cat f1 > f3
```

5 MANIPULAÇÃO DE FICHEIROS DE TEXTO

5.1 CONTAGEM DE LINHAS NUM FICHEIRO DE TEXTO

wc (*Word Count*) é o utilitário Linux que permite a contagem de linhas, palavras e caracteres num ficheiro de texto.

```
$ wc [ -cwl ] nome_dos_ficheiros
```

5.2 SUBSTITUIR CARACTERES NUM FICHEIRO DE TEXTO

O comando **tr** (*translate*) na sua forma mais simples permite substituir os caracteres especificados na *string_input* pelos especificados na *string_output*.

O comando **tr** é um filtro, lê o *standard input* (teclado) e devolve a saída para o *standard output* (monitor).

O *standard input* pode ser redireccionado para um ficheiro (<) ou fornecido por um *pipe* (|). O *standard output* também pode ser redireccionado (>), embora no Linux não seja permitido usar o mesmo ficheiro como entrada e saída de um comando, pode-se contornar esta limitação recorrendo a um ficheiro temporário.

Com a opção **-d**, os caracteres não são substituídos, mas removidos.

A opção **-s** remove os caracteres que se repetem, pelo que se torna muitas vezes útil para remover os duplos espaços, triplos, etc., mantendo apenas um espaço.

```
$ tr [-cds] string_input [string_output]
```

Alguns exemplos:

```
ls -l | tr -s " "      Transforma um conjunto de espaços na saída do comando ls -l para apenas um espaço.
```

```
ls -l | tr "a" "A"     Substitui o carácter "a" pelo carácter "A".
```

5.3 EXTRAIR COLUNAS NUM FICHEIRO DE TEXTO

O comando **cut** permite visualizar ficheiros como matrizes com linhas e colunas e permite extrair as colunas especificadas.

O comando **cut** pode analisar o ficheiro de duas formas: os ficheiros podem ser constituídos por colunas separadas por determinado carácter (por defeito espaço, mas é configurável através da opção **-d**). Para

extrair colunas a este tipo de ficheiros tabulares usa-se **-f** seguido de uma lista de colunas a extrair (por exemplo **-f1,4,3**).

A opção **-s** obriga o comando **cut** a suprimir linhas do ficheiro que não contenham o caracter delimitador.

A opção **-c** permite cortar um conjunto de caracteres especificando, para isso, a sua posição na linha, por exemplo **cut -c1**, corta apenas o primeiro caracter de cada linha.

```
$ cut -flista [-dchar] [-s] [ficheiro]
$ cut -clista [ficheiro]
```

Alguns exemplos:

```
ls -l | tr -s " " | cut -f1,4 -d" " |
```

Transforma um conjunto de espaços na saída do comando **ls -l** para apenas um espaço e de seguida vai extrair as colunas 1 e 4 que são delimitas por um " ".

```
cat fich | cut -c0-3
```

Obtém como resultado os três primeiro caracteres de cada linha.

5.4 "COLAR" FICHEIROS

Muitas vezes existe a necessidade de cortar várias colunas a um ficheiro para depois as voltar a colar noutro com numa ordem diferente. O utilitário Linux **paste** lê um ou mais ficheiros e cola as colunas correspondentes pela ordem pretendida.

```
$ paste [-s] [-dlista] [ficheiros]
```

Por defeito as colunas são separadas por espaço, mas podem ser separadas por qualquer caracter especificado com a opção **-d**, esta opção permite a utilização de uma lista de caracteres, sendo que eles são usados ciclicamente (1º, 2º, 3º, 1º, 2º, ...).

A opção **-s** permite juntar informação de vários ficheiros numa linha e não em colunas.

O comando **paste** não é um filtro. É necessário indicar na linha de comandos os ficheiros a usar, porque o utilitário não lê o *standard input*. Se em alguma situação houver a necessidade de usar o **paste** como filtro, deve escrever-se um hífen (-) na linha de comandos em vez do nome dos ficheiros.

Alguns exemplos:

```
who | cut -f1 -d" " | paste -s -
```

Apresenta os utilizadores activos numa única linha.

```
$ls -l | tr -s " " | cut -f1,2 -d" " > f1
```

Primeiro são criados os ficheiro **f1** e **f2** que contem as colunas 1,2 e 3,4 da saída do comando **ls -l**. De seguida o comando **paste** junta as colunas dos 2 ficheiros separa-os pelo carácter ":".

```
$ls -l | tr -s " " | cut -f3,4 -d" " > f2
$paste -d: f1 f2
```

6 COMANDOS AVANÇADOS (O USO DE EXPRESSÕES REGULARES)

6.1 INTRODUÇÃO ÀS EXPRESSÕES REGULARES

Vários comandos em Linux usam expressões regulares para efectuar operações de procura (*searching*) e de compatibilidade de padrões (*pattern matching*).

grep (*Global Regular Expression Printer*) é um comando para encontrar linhas num ficheiro de acordo com padrões específicos. Um padrão que seja usado para procurar num ficheiro uma dada sequência de caracteres é designado por **expressão regular**.

Em conjunto com o redireccionamento de *input* e *output*, bem como com outras funcionalidades de manipulação de ficheiros, o comando **grep** é extremamente poderoso: não só podemos procurar a próxima ocorrência de uma palavra mas também a próxima ocorrência de uma palavra no início ou fim de uma linha, etc.

Para que tal ocorra, é forçoso que certos caracteres tenham um significado especial se usados no interior de uma expressão regular. Claro que tem de ser possível a utilização do significado literal desses caracteres, tal como se fez na *shell* retirando-lhes, se necessário, o seu significado especial.

Qualquer outro carácter no interior de uma expressão regular que não seja um carácter especial é tomado literalmente.

6.2 O COMANDO GREP

O funcionamento do comando **grep** é simples de compreender: percorre as linhas de um ou mais ficheiros, procurando aquelas que contêm a expressão regular; se encontrar imprime toda essa linha (se nenhum ficheiro for indicado o comando lê o *standard input*).

Naturalmente é um comando que pode ser usado num *pipeline*.

Grande parte dos caracteres que têm um significado especial se usados numa expressão regular, também têm significado especial para a *shell*, pelo que para impedir que esta os interprete, se colocam entre quotas (' ').

A sintaxe dos comando **grep** é a seguinte:

```
grep [-vxcilnbs] [-f ficheiro] [-e] 'expressão regular' [ficheiro ...]
```

Os caracteres que têm significado especial quando usados em expressões regulares são vários, e listados na tabela seguinte. Para retirar-lhes o significado especial, basta serem precedidos por ****, por exemplo.

^	Procura os caracteres no início de uma linha
\$	Procura os caracteres no fim de uma linha
[lista]	Procura apenas um carácter da lista entre parêntesis
.	Apenas um carácter
*	Repete a expressão regular anterior 0 ou mais vezes.
\{n,m\}	Repete a expressão regular de um carácter n ou mais vezes mas não mais de m vezes
\{n\}	Repete a expressão regular de um carácter exactamente n vezes
\{n,\}	Repete a expressão regular de um carácter n ou mais vezes
*	Abreviatura de \{0,\}
+	Abreviatura de \{1,\}
?	Abreviatura de \{0,1\}
 	OR lógico de expressões regulares
()	AND lógico de expressões regulares

[^ ...]	A negação dos caracteres da lista
---------	-----------------------------------

Exemplos:

grep '^No' fortunes	linhas que começam com No no ficheiro fortunes
grep '^i0' /etc/passwd	linhas que começam por i0 no ficheiro /etc/passwd
grep '/bin/sh\$' /etc/passwd	linhas que acabam em /bin/sh no ficheiro /etc/passwd
grep '[Dd]isk' fortunes	linhas que contêm Disk ou disk no ficheiro fortunes
grep '^i0[2-3]0' /etc/passwd	linhas que começam por i020 ou i030 no ficheiro /etc/passwd
grep '^i0[^3-7]9' /etc/passwd	linhas que começam por i009,i019,i029,i089 ou i099 no ficheiro ...
grep '[Uu]...' fortunes	linhas que tem palavras de 4 caracteres em que o primeiro é U ou u
grep '[Vv][a-z]*' fortunes	linhas que têm palavras que começam por V ou v, seguidos de 0 ou mais letras minúsculas
grep -E '[FZ][d-w]+' fortunes	linhas que contenham palavras começadas por maiuscula F ou Z seguidas de uma ou mais minúsculas entre d e w.
grep -E '^'[AEIOU][^aei\]\\{1,3\\}'	linhas que começam por uma vogal maiúscula, seguidas de uma até três ocorrências de minúsculas que não sejam as vogais {"a", "e", "i"}.
grep -E 'i0(20 30)5' /etc/passwd	linhas que contêm i0205 Ou i0305 no ficheiro /etc/passwd

6.2.1 OPÇÕES DO COMANDO grep

-c	Imprime apenas o nº de linhas que encontra
-v	Inverte a procura. Mostra as linhas que não têm a expressão regular.
-i	Ignora diferenças entre maiúsculas e minúsculas.
-n	Precede a linha de resultado com o respectivo número.
-x	Só para o fgrep. Opção exacta: só imprime se a string for exactamente igual.
-b	Precede a linha de resultado com o nº do bloco de disco (administração).
-s	Só pra grep. Suprime as mensagens de erro.
-f	Lê de um ficheiro e não de stdin.
-l	Lista os ficheiros e não as linhas que contêm a expressão regular.
-e	Permite indicar uma expressão regular que comece com hífen (-)

6.3 O COMANDO find

Aviso: existem versões do **find** para outros sistemas operativos. O presente texto refere-se ao **find** da GNU versão 4.1. Outras versões do comando **find** podem não possuir as mesmas funcionalidades. Este

documento não é de forma alguma exaustivo, e se quer mesmo saber como o **find** funciona, nada melhor do que fazer: **man find**

O comando **find** serve para encontrar ficheiros, o exemplo de utilização mais simples é:

```
find
```

Este comando encontra todos os ficheiros existentes debaixo do directório corrente, e imprime o seu nome (acção por defeito). Se existirem, os primeiros argumentos do comando **find** são sempre os directórios nos quais queremos procurar os ficheiros.

No entanto a maior utilidade do comando **find** vem do facto de podermos seleccionar os ficheiros que queremos de muitas formas segundo as várias opções.

Para encontrar ficheiros com um determinado nome utiliza-se a opção **-name**:

```
-name nome_do_ficheiro
-iname nome_do_ficheiro      ( não distingue entre maiúsculas e minúsculas )
```

Exemplos:

```
$find -name core
find . -name \*.c
```

O primeiro exemplo procura ficheiros com nome **core** e o segundo vai procurar ficheiros com o nome **"*.c"**, note-se que a barra **"\"** vai impedir a shell de fazer a expansão do metacaracter **"*"**. Neste caso quem faz a expansão do metacaracter **"*"** é o próprio comando **grep**. Em alternativa pode-se delimitar a expressão por peliças **' '**.

É também possível encontrar ficheiros com um determinado path:

```
-path      pathname
-ipath     pathname      ( não distingue entre maiúsculas e minúsculas )
```

Exemplo:

```
find . -path misc      Pode encontrar ( se existirem ) os ficheiros:
                        ./misc.c
                        ./src/misc/test.c
                        ./src/prog1/misc3.c
```

Encontrar ficheiro de um certo tipo:

```
-type tipo
```

O tipo poder ser:

b <i>block device</i>	f ficheiro "normal"
c <i>char device</i>	p <i>named pipe</i>
d <i>directório</i>	s <i>socket</i>
l <i>link simbólico</i>	

Exemplos:

```
find . -type d
find . -type f
```

Encontrar ficheiros de um determinado grupo ou utilizador:

```
-user username
-uid user_id
-group nome_do_grupo
-gid group_id
-nouser          não pertencem a nenhum utilizador (já não existe o utilizador)
-nogroup         não pertencem a nenhum grupo (já não existe o grupo)
```

Exemplos:

```
find . -username joe
find . -uid 1024
find . -nouser -print
```

Encontrar ficheiros de um determinado tamanho:

```
-size n
```

Nota: o tamanho é escrito em blocos, uma unidade que pode não ser igual em todos os sistemas Linux. Para evitar problemas podem usar-se os seguintes sufixos:

c – chars=bytes
k – kbytes
M – Mbytes

Exemplos:

```
find . -size 100          ficheiros com 100 blocos
find . -size +100c        ficheiros com mais de 100 bytes
find . -size -100k        ficheiros com menos de 100 kbytes
```

Encontrar ficheiros com uma certa data/tempo:

```
Data de acesso dos ficheiros
  -atime -amin -anewer
Data de "mudança" (change) do ficheiro (pode ser mudança de dono ou
atributos !!! )
Nota: time - dias  min - minutos
  -ctime -cmin -cnewer
Data de modificação (do conteúdo) do ficheiro
```

```
-mtime -mmin -newer
```

Exemplos:

```
find . -atime 2      ficheiros acedidos há dois dias
find . -amin 1       ficheiros acedidos há um minuto
find . -mtime -5     ficheiros modificados há menos de 5 dias
find . -mmin +30     ficheiros modificados há mais de 30 minutos
find . -newer fich1.c ficheiros modif. Depois do ficheiro fich1.c
Ficheiros com certas permissões (exactamente)      -perm n
Ficheiros com certos bits das permissões todos activados -perm -n
Ficheiros com certos bits das permissões activados  -perm +n
```

A opção `perm` permite encontrar ficheiros com determinadas permissões (as permissões são escritas em octal):

```
find . -perm 700      ficheiros com as permissões a 700      (ficheiros com rwx----- apenas! )
```

Podemos usar certas opções antes dos comandos:

```
-daystart      medir os dias desde o inicio do dia e não desde há 24 horas
-xdev          excluir outros "devices" ( serve para excluir outros discos ou servidores da rede )
-mount        excluir outros "devices" ( o mesmo que -xdev para compatibilidade )
-maxdepth n    descer no máximo n níveis de subdirectórios
-mindepth n    considerar apenas os ficheiros a n ou mais níveis de subdirectórios
-follow        seguir links simbólicos
```

Exemplos:

```
find . -mtime 1      ficheiros modificados entre 24 e 48 horas atrás
find . -daystart -mtime 1  ficheiros modificados ontem
```

Podemos usar mais do que uma opção:

```
find . -xdev -mindepth 3 -name core
```

ou podemos usar mais do que uma condição:

```
find . -xdev -user joe -name core
```

Por defeito, o `find` faz o "*and*" das condições, ou seja, o comando só é executado se todas as condições se verificarem, embora se possam usar outros operadores lógicos

```
-a      -and
```



```
-o      -or
-not    !
```

e podemos usar parêntesis para agrupar as condições (protegidos da shell).

Nota: a avaliação das expressões é feita em "curto-circuito", isto é se num *“or”* a primeira expressão é verdadeira, ou num *and* a primeira expressão é falsa, a segunda expressão não é avaliada.

Exemplos:

```
find . -user joe -perm -002
find . -user joe -a -perm -002
find . -nouser -o -nogroup -o \( -name core -a -type f \)
```

Por último podemos executar comandos nos ficheiros que escolhemos com:

```
-exec comando {} \;      executar um comando
-ok  comando  {} \;      executar um comando condicionalmente(pergunta ok?)
```

O **find** substitui **{}** pelo nome de cada ficheiro que vai encontrando e o **\;** serve para dizermos ao **find** onde acaba o nosso comando.

Exemplos:

```
find / -user joe -exec rm -rf {} \;      apagar todos os ficheiros do utilizador joe
find . -type f -name \*.bak -ok rm {} \;  apagar condicionalmente os ficheiros *.bak
```

Nota:

Nalgumas *shells* convém fazer o escape das chavetas, para saber em quais, experimentar ou *rtfm*. Porque é que temos de assinalar o fim de um comando? Porque as acções para o **find** comportam-se como condições e podemos ter mais de uma acção no mesmo **find**.

Exemplo:

```
find . -name \*.c -exec cp {} ~/backup \; -size +10k -ok rm {} \;
```

Encontra todos os ficheiros ***.c** , copia-los todos para o directório **~/backup** e perguntando se o utilizador quer apagar os que têm mais de 10 kbytes.

7 Controlo de Processos em LINUX

7.1 *Background vs Foreground*

No processamento em *foreground* a shell (processo pai) espera que o comando (processo filho) seja executado antes de exibir um novo prompt.

Pelo contrário o processamento em *background* permite que sejam executados vários processos simultaneamente e que novos comandos sejam iniciados sem que os anteriores tenham terminado a sua execução.

7.2 *Executar Processos em Background; PIDs*

Para executar um comando/programa em *background* basta colocar o caracter **&** no fim da linha de comandos.

O Process ID Number é a resposta da shell à linha de comandos terminada em **&**. Pode ser visto como o bilhete de identidade do processo a correr em *background*.

```
$ find /bin /etc -name passwd 2>/dev/null &
12283
$
```

7.3 *Standard Input, Standard Output e Standard Error*

Executar um comando em *background* não redirecciona automaticamente o standard input, o standard output e o standard error. É contudo aconselhável que o utilizador o faça pelas razões que a seguir se apresentam.

Se o *stdin* não for redireccionado, tanto a shell como o comando vão estar à espera de ler o teclado e não há possibilidade de se saber que caracteres são entrada do comando e que caracteres são uma nova linha de comandos.

Se o *stdout* ou o *stderr* não forem redireccionados para um ficheiro, o utilizador será interrompido pelo resultado da execução dos comandos e por possíveis mensagens de erro, enquanto pretende escrever outras linhas de comandos, o que causará uma enorme confusão.

Assim, aconselha-se que para lançar processos em background se recorra a:

```
$ (linha de comandos < ficheiro_in > ficheiro_out) >2 /dev/null &
```

Enviar dados para */dev/null* é equivalente a deitar os dados ao “lixo”.

7.4 *O comando Wait*

O comando `wait` obriga a shell a esperar a execução de todos os comandos em background, antes de exibir uma nova prompt. Assim se se invocar este comando a prompt da shell só será apresentada depois de todos os comandos terminarem.

```
$ wait
```

7.5 *O comando PS - Process Status*

O comando `ps` permite ao utilizador saber o estado de um processo. A sua sintaxe genérica é:

```
$ ps [-lef] [-t lista_terminais][-u lista_utilizadores][-p lista_processos]
```

Opções:

- l – long list
- f – full list
- e – environment

Na sua forma mais simples, isto é sem argumentos, permite ao utilizador ver o estado dos seus processos (incluindo os que estão em background).

```
$ alvega> ps
  PID TTY          TIME CMD
 23455 pts/0    00:00:00 csh
 23472 pts/0    00:00:00 ps
alvega>
```

```
alvega> ps -l
  F S  UID  PID  PPID  C PRI  NI ADDR  SZ WCHAN  TTY          TIME CMD
 100 S  1017 23455 23454  0  71   0  -  1089 rt_sig pts/0    00:00:00 csh
 000 R  1017 23479 23455  0  78   0  -  1489 -      pts/0    00:00:00 ps
alvega>
```

Onde:

- F - mostra as *flags* associadas aos processo. As *flags* aparecem em octal;
- S – mostra o estado corrente do processo (S-sleeping, W-waiting, R-running, ...);
- UID – *User Id* do dono do processo;
- PID – *Process Id*;
- PPID – *Pid* do processo pai deste comando;
- C – dá indicação sobre os recursos centrais atribuídos a este processo;
- PRI – prioridade do processo (0-máxima);
- NI – *nice value* (descrito em 9.7);
- ADDR – endereço de memória primária onde reside o processo (caso ele se encontre em memória primária);
- SZ – tamanho da imagem do processo;
- WCHAN – informação sobre um evento que tenha colocado o processo em estado de *waiting* ou *sleeping*;
- TTY – identificação do terminal que lançou o processo;
- TIME – tempo de execução do processo;
- CDM – nome do comando executado.

7.6 O comando Kill

O comando `kill` é usado para enviar sinais para o processo em *background*.

Um sinal é uma mensagem simples que avisa o processo da ocorrência de algum evento anormal. Os sinais podem causar:

- terminar o processo;
- suspender o processo;

- não fazer nada.

A sua sintaxe genérica é:

```
$ Kill [-l] [sinal] pid
```

A opção `-l` apenas funciona como "help" e devolve a lista de todos os sinais do sistema. Os sinais apresentados abaixo, têm por função terminar o processo. Se não for especificado nenhum sinal é assumido que se envia o sinal 15 (TERM).

```
1 -   HUP    - Hangup
9 -   KILL   - termina o processo
15-  TERM   - Default
```

```
alvega> kill -l
HUP INT QUIT ILL TRAP ABRT BUS FPE KILL USR1 SEGV USR2 PIPE ALRM TERM
STKFLT
CHLD CONT STOP TSTP TTIN TTOU URG XCPU XFSZ VTALRM PROF WINCH POLL PWR SYS
RTMIN RTMIN+1 RTMIN+2 RTMIN+3 RTMAX-3 RTMAX-2 RTMAX-1 RTMAX
alvega>
```

7.7 Prioridade de um Processo

A prioridade de um processo é calculada pelo sistema operativo e determina qual o processo que deve ser executado a seguir. O cálculo é feito com base na utilização da UCP e do "*nice value*".

O *nice value* é um número de 0 a 20. Onde o **0** é só um bocadinho simpático e **20** é realmente simpático.

A prioridade do processo não pode ser alterada, mas podemos afectar o seu *nice value* com o comando **nice**.

O comando **nice** é vulgarmente utilizado em processos a correr em *background*, mas também pode ser usado para processos em *foreground*.

```
$ nice [-n°] linha de comandos [&]
```