Conceitos básicos de *Shell* de `UNIX`

Departamento de Ciência de Computadores
Faculdade de Ciências, Universidade do Porto

Versão 0.3: Outubro 2012

Conteúdo

Uma *shell* é um interpretador de linguagens de comandos que permite executar comandos interactivamente ou dum ficheiro. Exemplos de *shells* em `UNIX` são a Bourne shell (`sh`), C shell (`csh`) ou Bourne-again shell (`bash`).

A forma geral de excutar comandos é:

```
% comando [-opcoes] [argumentos]
```

Para a generalidade dos comandos pode-se utilizar o *manual de ajuda on-line*:

```
% man [n] comando
```

onde "n" refere-se à seção do comando (Unix organiza os comandos em conjuntos dependendo de sua funcionalidade e nível de utilização: user-level, library-level, system-level etc). Os parênteses retos são apenas para indicar que o "n" é opcional. Por exemplo, se quisermos saber mais informação sobre a `bash`, podemos simplesmente digitar: `man bash`

1 Ficheiros e Diretórios

1.1 Ficheiros

Um *ficheiro* é uma abstração que permite manipular e organizar a informação guardada em memórias secundárias (discos, etc) e outros recursos. O `UNIX` trata os periféricos (impressoras, terminais, etc) também como ficheiros (especiais).

Um ficheiro possui pelo menos as seguintes características:

- tem um nome
- não é volátil: não desaparece quando o computador é desligado
- pode ler e escrever informação
- pode ser criado, copiado, apagado, duplicado

Em `UNIX`, um ficheiro é uma sequência de *bytes* com ainda:

- data da criação, data da última utilização, data da última modificação etc
- número de bytes (tamanho)
- identificação do utilizador (proprietário do ficheiro)
- identificação do grupo
- permissões (por exemplo, de leitura, escrita, execução etc)

Os ficheiros podem ser:

ficheiros normais

- de texto
- binários
  - executáveis
  - codificação digital de imagens
  - codificação digital de som
  - etc...

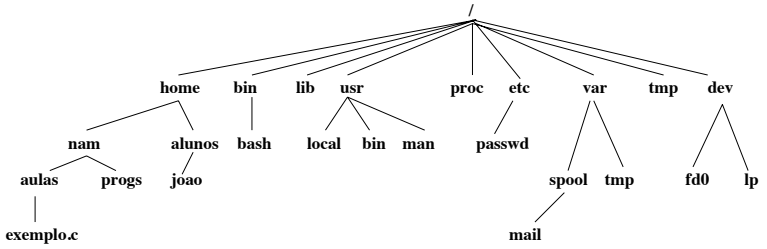
diretórios conjuntos de ficheiros

especiais

- de caracteres (impressoras, etc)
- de blocos (discos duros)
- *pipes* (encadeamento de canais)
- *sockets* (comunicação entre processos)

1.2 Diretórios

Os ficheiros organizam-se em *diretórios*. Um diretório é um tipo especial de ficheiro onde se guarda informação sobre outros ficheiros. Em particular, outros diretórios, obtendo assim uma organização hierárquica em árvore:



O diretório `/` (*root*) é a raiz da árvore.

Comando para manipulação e navegação na árvore de diretórios

Comando	Exemplo	Descrição
`mkdir dir`	cria o diretório `dir`	`mkdir aulas`
`pwd`	escreve o nome do diretório corrente	`pwd`
`cd dir`	muda o diretório corrente para `dir`	`cd aulas`
`ls dir`	lista o conteúdo do diretório `dir`	`ls aulas`
`rmdir dir`	remove o diretório `dir`, que deve existir e estar vazio	`rmdir aulas`

Comandos para manipulação de ficheiros

Comando	Descrição	Exemplo
`cat fich1 ... fichn`	escreve sucessivamente o conteúdo dos ficheiros `fich1`, ..., `fichn`	`cat aula1 aula2` na saída padrão (ecrã)
`mv fich nome`	se `nome` for um diretório existente, então coloca o ficheiro `fich` no diretório `nome`; se `nome` for um ficheiro existente, então o ficheiro `fich` passa a chamar-se `nome` e substitui o ficheiro `nome` anterior; se não existir nenhum diretório ou ficheiro `nome`, então o ficheiro `fich` passa a chamar-se `nome`	`mv aula1 aulas`
`mv dir1 dir2`	coloca o diretório `dir1` dentro do diretório `dir2`, caso este exista; caso contrário, o diretório `dir1` passa a chamar-se `dir2`.	`mv ic aulas`
`cp fich nome`	análogo ao comando `mv`, mas mantém uma cópia do ficheiro `fich` original	`cp aula2 aulas`
`rm fich1 ... fichn`	remove os ficheiros `fich1`, ..., `fichn` do diretório corrente	`rm aula2`

1.3 Nomes Completos e Relativos. Caminhos

Os ficheiros podem ser identificados indicando a sua localização na árvore em relação ao diretório raiz "/". Para tal escrevem-se os nomes dos diretórios, que estão no caminho desde a raiz, separados por "/"e seguidos do nome do ficheiro. Obtém-se o *nome completo* (ou absoluto). Por exemplo, o nome completo do ficheiro `exemplo.c` é: `/home/prof/nam/progs/exemplo.c`.

No entanto, não é muito cómodo ter de referir sempre o nome completo de um ficheiro ou diretório. Assim, a cada processo está associado o `diretório corrente de trabalho`: i.e, o diretório em que se "está"em cada momento. Representa-se por ".". Se o nome de um ficheiro não começar por "/", então a sua localização vai ser determinada em relação ao *diretório corrente*. O caminho desde o diretório corrente, é o que se designa por *nome relativo* de um ficheiro. Se o diretório corrente for `/home/nam/`, o nome relativo do ficheiro `exemplo.c` é: `progs/exemplo.c` ou (`./progs/exemplo.c`)

E como referir o nome relativo do diretório `alunos`? Esse diretório não está na mesma subárvore de "."Para podermos "subir"na árvore, referimo-nos por ".." ao *diretório pai* do diretório corrente. E "../.."ao diretório pai do diretório pai do diretório corrente, etc.

O *diretório casa do utilizador* é o diretório que lhe é atribuido quando a conta é criada. É o diretório corrente da *shell*, quando um utilizador entra no sistema (`login`). Representa-se por `~nome_do_utilizador`. Exemplo: `~nam` pode ser `/home/nam`.

Nomes de ficheiros	Descrição	Exemplo
`/dir1/···/dirn-1/dirn/nome`	designa o caminho (absoluto) a partir da raíz para o ficheiro ou o diretório nome que se encontra no diretório `dirn`, que está dentro do diretório `dirn-1`, ..., que está dentro do diretório `dir1`, que está na raíz.	`/usr/sbb/aulas/ic`
`..`	designa o diretório acima na hierarquia	`../../pi1`
`.`	designa o diretório corrente	`./teste`
`~`	designa o diretório casa	`~/aulas/ic`
`~nome`	designa o diretório casa do utilizador `nome`	`~nam`

1.4 Permissões

Tanto os ficheiros como os processos em `UNIX` têm um sistema de permissões de acesso. Cada ficheiro (ou processo) pertence sempre a um utilizador. Um processo (que pertence a um utilizador) só pode aceder a um ficheiro se as permissões desse ficheiro o autorizarem. Por outro lado, os utilizadores organizam-se em grupos. As permissões podem ser para:

`u:` utilizador proprietário

`g:` todos os utilizadores do grupo do utilizador

`o:` outros utilizadores

`a:` todos os utilizadores (soma dos anteriores)

As permissões existem para ficheiros e diretórios.
O tipo de permissão pode ser:

Permissão	Ficheiros	Diretórios	Quem	Octal
r	ler o contéudo	listar o conteúdo do diretório	u g o	400 040 004
w	escrever	adicionar/copiar/remover ficheiros	u g o	200 020 002
x	execução	aceder a ficheiros e subdiretórios dentro do diretório	u g o	100 010 001

As permissões para o *utilizador*, *grupo* e *outros* agrupam-se em sequências de 3 caracteres correspondendo, respectivamente a *leitura*, *escrita* e *execução*, e onde `-` indica a ausência duma dada permissão. Por exemplo um ficheiro com a sequência:

```
rw-r--r-
```

permite a leitura e escrita para o utilizador, e só leitura para grupo e outros.

Da esquerda para a direita, o primeiro grupo de 3 caracteres corresponde às permissões para o utilizador (`u`, `rw-`), o segundo grupo, corresponde às permissões do grupo (`g`, `r--`) e o terceiro corresponde às permissões para outros utilizadores (`o`, `r--`).

Comandos que alteram as permissões

Comando	Exemplo	Descrição
`chmod` modo `args`	`chmod o-w ola`	permite mudar as permissões (neste caso, outros utilizadores (o) perdem a permissão de escrita sobre o ficheiro ola
`chown`	`chown nam ex.c`	permite o *super-user* mudar o proprietário dum ficheiro

Modos combinados (exemplos): somando as permissões

777	tudo é permitido para todos
700	só o utilizador tem todas as permissões
000	nada para ninguém...
755	só o utilizador pode adicionar/remover
644	só o utilizador pode escrever
600	só o utilizador pode ler e escrever
666	todos podem ler e escrever

Exemplo 1 *Mudando permissões de um ficheiro*

```
% ls -l
-rw-r--r--   1 nam      nam      10 Oct 17 21:23 ola
drwx------   6 nam      nam    2048 Sep 27 16:03 aulas/
drwxr-xr-x   2 nam      nam    1024 Oct 17 20:27 teste/

%chmod ug-r ola
% ls -l ola
--w----r--   1 nam      nam      10 Oct 17 21:23 ola

% chmod a+w ola
% ls -l
-rw-rw-rw-   1 nam      nam      10 Oct 17 21:23 ola

% chmod 000 ola
% ls -l ola
----------   1 nam      nam      10 Oct 17 21:23 ola

% rm ola
rm: remove write-protected file ‘ola’? n

%chmod 741 ola
%ls -l ola
-rwxr----x   1 nam      nam      10 Oct 17 21:23 ola*

%chmod 644 ola
% ls -l ola
-rw-r--r--   1 nam      nam      10 Oct 17 21:23 ola

%chmod 777
%ls -l ola
-rwxrwxrwx   1 nam      nam      10 Oct 17 21:23 ola*

%file aulas
aulas: directory
```

```
%file  ex1.c
ex1.c: C program text
```

2 Processos

Os processos em UNIX organizam-se em *árvore*. O comando pstree do UNIX, permite visualizar a árvore de processos do sistema num dado instante.

```
init-+-apache---9*[apache]
   |-apmd
   |-automount
   |-bash
   |-cron
   |-5*[getty]
   |-gpm
   |-inetd---nmbd
   |-klogd
   |-kpiod
   |-kswapd
   |-rwhod
   |-sshd
   |-syslogd
   |-wdm-+-XBF_NeoMagic
   |     `-wdm---WindowMaker-+-communicator-sm---communicator-sm
   |                         |-emacs---xdvi---xdvi.bin---gs
   |                         |-exmh-+-ispell
   |                         |      `-wish
   |                         |-wmtime
   |                         |-xterm---bash---bash
   |                         `-xterm---bash---pstree
   |-xconsole.real
   `-xfs
```

O commando ps fornece informação sobre os processos em curso.

```
 ps
PID TTY STAT TIME COMMAND
154   1 S    0:00 -bash
169   1 S    0:00 xinit /usr/X11R6/lib/X11/xinit/xinitrc
176   1 S    0:06 /usr/X11R6/bin/WindowMaker
204   1 S    0:50 emacs
430  p0 S    0:01 xdvi.bin -name xdvi sliic98.dvi
512   1 S    0:00 /usr/bin/X11/xfig
522  p1 S    0:00 /bin/bash -login
534  p1 R    0:00 ps
```

Alguns comandos que manipulam processos

Comando	Descrição	Exemplo
`sleep seg`	permite que um processo fique suspenso por `seg` segundos e depois se reactive	`%sleep 10`
`wait [pid]`	Uma `shell` fica á espera que um processo filho `pid` termine (ou todos)	`wait 204`
`kill [-sinal] pid`	Envia sinais a processos	`kill -9 204`
`top`	permite uma monitorização interactiva dos processos.	

Os *sinais* são números que são associados a acções sobre processos.

Sinal	Valor	Acção	Comentário
SIGHUP	1	Terminação	Desconexão de um terminal
SIGINT	2	Terminação	Interrupção do teclado
SIGILL	4	Terminação	Instrução ilegal
SIGKILL	9	Terminação, nunca é ignorado	Sinal de matar

3 Funcionalidades duma *shell*

Uma *shell* inclui normalmente:

- comandos internos (*buit-in*). Ex: `kill`, `echo`, `logout`, `pwd`, `cd`, `help`, etc
- expansão de nome de ficheiros (*wildcards*)
- variáveis locais e de ambiente
- redireção dos canais de entrada/saída
- encadeamento de processos (*pipes* `|`)
- execução atrás (*background* `&`)
- ficheiros de comandos (*scripts*)
- sequências de comandos
- controlo de fluxo
- sub-shells. Ex: agrupamento '`(` '`)`', execução de *scripts*
- substituição de comandos
- meta-caracteres

Invocação duma `shell` Quando uma *shell* é invocada, em modo interactivo, depois de um *login* ou pelo comando associado:

1. lê um ficheiro inicial de configuração, normalmente localizado no diretório casa do utilizador ou no diretório `/etc/`. Ex: `.profile, .bashrc, /etc/profile`
2. mostra um ou mais caracteres, que se designam por *prompt*, (p.e. `%`) e fica à espera de comandos do utilizador que terminam com a mudança de linha (tecla `Enter`)
3. Uma linha de comando consiste numa ou mais palavras separadas por espaços. A primeira é o nome do comando e as restantes, se existirem são argumentos ou opções. Se um comando ocupar mais que uma linha, usar o caracter `\`
4. Se o comando for `control-d` ou `exit`, isso significa `fim-de-dados` e a *shell* termina; caso contrário executa o comando indicado e volta ao passo 2.

Vamos considerar em mais pormenor algumas das funcionalidades da `bash` (ver também `man bash`).

3.1 Expansão de nomes de ficheiros (*glob*)

Permitem referir vários ficheiros cujo nome verifica um determinado padrão. Podem ser usados em qualquer comando cujo argumento possa ser uma lista de ficheiros.

Comando	Descrição	Exemplo
`*`	substitui qualquer sequência de zero ou mais caracteres	`ls -l *.c`
`?`	substitui um qualquer caracter	`ls -l a?`
`[...]`	qualquer caracter da lista	`ls -l ex*.[ch]`
	qualquer caracter entre dois caracteres separados por `-`	`ls -l ex[1-9].c`
`[^...]`	caracteres que não pertencem à lista	`ls -l ex[^12].c`
`{...}`	conjuntos alternativos	`ls -l *.{gif,jpg}`

Exemplo 2 *Expansão de nomes de ficheiros*

```
% ls *
a.c  ex1.c  ex1.h  ex2.c  ex2.h  ex3.c  ex4.c exa.c
% ls *.c
a.c  ex1.c  ex2.c  ex3.c  ex4.c exa.c
% ls ex?.c
ex1.c  ex2.c  ex3.c  ex4.c exa.c
% ls ex*.[ch]
ex1.c  ex1.h  ex2.c  ex2.h  ex3.c  ex4.c exa.c
% ls ex[1-9].c
ex1.c  ex2.c  ex3.c  ex4.c
% ls ex[^12].c
ex3.c  ex4.c exa.c
```

Exemplo 3 *Listar ficheiros (ou diretórios) cujo nome:*

- *começa por uma letra:* `ls -l [A-Za-z]*`
- *tem apenas 3 caracteres:* `ls -l ???`
- *não terminam em c:* `ls -l *[^c]`
- *terminam em* `txt` *ou* `d`*:* `ls -l *{txt,d}`

3.2 Edição na linha de comandos

Registo de comandos

Comando	Descrição
`history`	produz a lista do registo de comandos
`history n`	produz a lista dos últimos n comandos no registo de comandos
`Ctrl-p`	para obter o comando anterior no registo de comandos (em alternativa pode utilizar a seta para cima)
`Ctrl-r` seq	permite procurar comandos no registo de comandos em que aparece a subsequência seq

Movimentos básicos do cursor

Comando	Descrição
`Ctrl-a` ou `HOME`	vai para o princípio da linha de comandos
`Ctrl-e` ou `END`	vai para o fim da linha de comandos

3.3 Variáveis e ambiente de execução

Uma *variável* permite associar um nome a outra sequência de caracteres. Existem variáveis pré-definidas e internas mas podem-se criar variáveis novas: `nome_de_variavel=valor`. Mas para obter o *valor* duma variável, temos que usar: `$nome_de_variavel`. Por exemplo:

```
% trab=/home/nam/aulas
% cd $trab
% pwd
/home/nam/aulas
```

Cada processo tem um ambiente de execução, que é constituído por um conjunto de variáveis. Algumas variáveis de ambiente pré-definidas, e normalmente inicializadas nos ficheiros de configuração:

`HOME`	nome completo do diretório casa do utilizador
`PATH`	lista de diretórios onde procurar ficheiros executáveis (comandos)
`USER`	identificação do utilizador
`SHELL`	a *shell* de *login*
`PS1`	caracteres de *prompt*
`PWD`	diretório corrente
`HOSTNAME`	nome da máquina

Algumas variáveis são inicializadas pela `shell`, outras são usadas por ela. Muitas aplicações têm variáveis de ambiente específicas. Um processo herda do pai as variáveis que forem exportadas (`export`):

```
% trab=/home/nam/aulas
% export trab
% bash
% echo $trab
/home/nam/aulas
% export DISPLAY=khayyam:0.0
```

As variáveis que não são exportadas, designam-se por *locais*. O comando `printenv` (ou `env`) escreve o valor das variáveis de ambiente:

```
% printenv
PWD=/home/nam/Aulas/ic
HOSTNAME=khayyam
MANPATH=/usr/local/man:/usr/man:/usr/X11/man:/usr/openwin/man
PS1=%
PS2=>
USER=nam
DISPLAY=:0.0
SHELL=/bin/bash
HOME=/home/nam
PATH=/home/nam/bin:/usr/sbin:/usr/bin:/bin:/usr/X11/bin:.
TERM=xterm-debian
_=/usr/bin/printenv
```

Para modificar estas variáveis, podemos usar o comando `export`:

```
export PATH=$PATH:/usr/games
```

Contudo, para que a modificação seja válida para outras invocações da `shell` terá que ser feita num ficheiro de configuração (por exemplo, `.bashrc` no diretório home).

Podemos também associar nomes a comandos. O comando `alias` (*buit-in*) permite definir (ou redefinir) comandos a partir doutros:

```
% alias ls="ls -l"
% alias la="ls -a"
% la
./
../
.bashrc
teste.c
aulas/
%
```

Podemos ainda *substituir* comandos. Um comando entre ‘ e ‘ é substituido pelo seu resultado (que é enviado para o canal de saída padrão): ‘`comando`‘ ou `$(comando)`.

Exemplo 4 *Variáveis e substituição*

```
%echo hoje e dia ‘date‘
hoje e dia Sun Oct 24 23:13:56 GMT 1999
```

ou

```
% aqui=‘pwd‘
% echo $aqui
/home/nam/aulas/
```

Resumo

Comando	Sintaxe	Exemplo
Definição	nome=valor	`A=255`
Substituição variáveis	`$nome`	`echo A=$A`
Substituição comandos	`‘comando‘`	`dir=‘ pwd ‘`
Ambiente	`env`	`env`
Alterar ambiente	`export nome=valor`	`export PATH=$PATH:~/outro_dir`

3.4 Redireção e Encadeamento

Cada processo tem pelo menos 3 canais de ligação (ficheiros):

Canal	Id	Usualmente	Denominação (em C)	Função
entrada	0	teclado	`stdin`	recebe dados
saída	1	ecrã	`stdout`	envia resultados
erro	2	ecrã	`stderr`	envia mensagens de erro

Se nenhum ficheiro é especificado num comando, o `stdin` é muitas vezes usado como entrada. Chama-se *filtro* um comando que (sem argumentos) lê o `stdin` e escreve no `stdout`.

Exemplo 5 *Utilização da entrada padrão por alguns comandos*

```
% cat
isto e um teste
isto e um teste^D
%
```

```
% wc
isto
e
um
teste^D
      4       4      16
%
```

```
% sort
isto
e
um
teste^D
e
isto
teste
um
%
```

Podemos redirecionar os canais *standard* para outros ficheiros ou comandos. O *canal de saída* pode ser redirecionado por `% comando > nome_de_ficheiro`. O comando escreve no ficheiro o seu resultado (apagando o ficheiro se ele existir).

Exemplo 6 *Redireção dos canais de entrada e saída padrão*

```
% cat > alunos
luis goncalves
ana luisa costa
joao tavares
^D
% ls -l > lista
% diff ex1.c ex1.old > ex1.dif
% echo "ola mundo" > pois
```

Exercício 1 *Qual a diferença entre `cat alunos` e `cat > alunos`?*

Para redirecionar o *canal de entrada* usa-se: `% comando < nome_de_ficheiro`. Neste caso, `comando` lê do ficheiro em vez do `stdin`.

Exemplo 7 *Redireção dos canais de entrada e saída padrão*

```
% wc < alunos
      3       7      44
% sort < alunos > ord_alunos
% mail joao@fc.up.pt < mensagem
% ls -l > lista
% wc < lista
```

Exercício 2 *Qual a diferença entre `wc alunos`, `wc < alunos` e `wc > alunos`?*

O redirecionamento do canal de saída pode ser feito sem apagar o ficheiro, mas acrescentando informação: `% comando >> nome_de_ficheiro`. Se o ficheiro `nome_de_ficheiro` já existir, a informação produzida pelo comando é acrescentada no fim do ficheiro.

Exemplo 8 *Redireção dos canais de entrada e saída padrão*

```
%cat >> alunos
joao pedro antunes ^D
% cat alunos
luis goncalves
ana luisa costa
joao tavares
joao pedro antunes
% echo "adeus" >> pois
```

Finalmente, podemos redirecionar o canal de erro: `% comando 2> nome_de_ficheiro`.

Exemplo 9 *Redireção do canal de erro*

```
%man emacs  2> m1
%cat m1
Reformatting emacs(1), please wait...
%
```

Sequências, agrupamentos e encadeamento de comandos Vários comandos podem ser executados em *sequência*: `% comando ; comando`. Neste caso, os vários comandos são executados no ambiente da `shell` actual. Por exemplo:

Exemplo 10 *Sequência de comandos*

```
% date; cd; pwd;
%Sat Oct 21 17:15:47 GMT 2000
/home/nam
```

Podemos executá-los numa sub-shell e o ambiente da `shell` que os executa não é alterado. É chamado um *agrupamento*: `% (comando ; comando)`. Mas os comandos no agrupamento têm os mesmos canais de entrada/saída.

Exemplo 11 *Agrupamento de comandos*

```
% (date; cd; pwd)> saida
% cat saida
Sat Oct 21 17:18:08 GMT 2000
/home/nam
% pwd
/home/nam/Aulas
```

Mas, o canal de entrada de um processo pode ser o canal de saída de outro processo, i.e ser um *encadeamento* (*pipe*): `% comando | comando`.

Exemplo 12 *Encadeamento de comandos (*pipes*)*

```
% ls -l > lista ; wc < lista
     89     794    5975

% ls -l | wc
     89     794    5975

% ls | sort | lpr

% grep main ex1.c
ex1.c:main() {
% grep main ex1.c | wc
      1       1       7
% ls -l | grep ex | wc
%    1       1       6

% who | grep nam
```

Resumo Podemos resumir a redireção e composição (sequência e agrupamento) de comandos na seguinte tabela:

`... > ficheiro`	redireção do `stdout` para ficheiro
`... >> ficheiro`	redireção do `stdout` para ficheiro com acumulação de informação
`... 2> ficheiro`	redireção do `stderr` para ficheiro
`... < ficheiro`	redireção do `stdin` de ficheiro
`... ; ...`	sequência de comandos
`(... ; ...)`	agrupamento de comandos
`... | ...`	encadeamento do `stdout` de um comando para o `stdin` de outro comando

3.5 Utilitários para formatação de output

Objectivo	Comando	Parâmetros	Exemplo
Escrever	`echo`		`echo esta frase`
Paginar output	`more` `less`		`find ~ -name '.c' | more`
Ordenar ficheiros	`sort`	`-n` comparação numérica `-u` unica (retira repetições) `-r` inverte a ordenação	`ls -l | sort`
Extrair colunas	`cut`	-f num número do "campo" -d char caracter separador -c range colunas a ver	`cut -f 3 -d: /etc/passwd`
Início de ficheiros	`head`	`-n` número de linhas	`head -10 /etc/passwd`
Fim de ficheiros	`tail`	`-num` número de linhas -f actualização continua	

3.6 Os utilitários `wc`, `grep` e `find`

Objectivo	Comando	Parâmetros	Exemplo
Contar elementos de textos	`wc`	-c caracteres -w palavras -l linhas	`wc -l fich.txt`
Procurar padrões de caracteres em ficheiros	`grep`	-i ignorar capitalização -n numerar linhas -v inverter selecção -w apenas palavras completas	`grep -n main proc.c`
Procurar ficheiros	`find`	-name padrão -print -atime ndias -ctime ndias -anewer ficheiro -cnewer ficheiro -type tipo -exec comando {} \;	`find ~ -name '*.h' -print` `find aulas -name '*.c' -exec wc {} \; -print`

3.7 Controlo de execução (*job control*)

Normalmente quando um comando é executado por uma *shell*, esta espera que o comando termine. A *shell* fica em `wait` e o comando executa à frente (*foreground*). No entanto, teclando `ctrl-z` o comando pode ser *suspenso*.

Exemplo 13 *Suspensão de comandos*

```
% man ls
...
^z
% jobs
[1]+  Stopped                 man ls
% fg %1
man ls
```

A shell associa um número a cada comando (*job*) que é lançado (não é o `PID`). Um comando pode ser executado atrás (em *background*), se for seguido pelo caracter `&`: `% comando &`.

Exemplo 14 *Execução de comandos atrás (em* background*)*

```
% cp ficheiro_grande ficheiro_novo &
[1] 356
% xv instrucoes.gif &
[2] 357
% emacs &
[3] 389
% compress aulas.tex
^z
[4]+ Stopped compress aulas.tex
% bg %4
[4]+ compress aulas.tex &
%jobs
[1]+  Running     cp ficheiro_grande ficheiro_novo &
[2]+  Running     xv instrucoes.gif &
[3]+  Running     emacs &
[4]+  Running     compress aulas.tex &
%kill %2
[2]+  Terminated    xv instrucoes.gif
```

Existem os seguintes comandos para o controle de jobs (`job control`):

Comando da `bash`	Função
`jobs`	lista de *jobs* activos
`bg %n`	coloca o *job* `%n` a correr atrás
`fg %n`	coloca o *job* `%n` a correr à frente
`kill %n`	mata o *job* `%n`
`ctrl-z`	suspende o comando que está actualmente a correr à frente
`comando &`	executa o comando atrás
`exit`	termina a *shell* indicando se há processos a correr

Quando um comando é executado *atrás* convém redirecionar o `sdtout` (e o `stderr`) para um ficheiro, de modo a nao confundir os processos seguintes que corram à frente.

Exemplo 15 *Confusão causada pela execução simultânea de processos (jobs) que compartilham o ecrã*

```
%grep main *.c &
[1] 1549

% ls
euros.c:main() {
euros*   euros2.c  seno.c   tutor1.c
euros2.c:main() {
fact2.c:   main(){
seno.c:main() { /* tabela da função "seno" entre 0..90 graus */
tutor1.c:main() {
euros.c    fact2.c   tutor1*  tutor2*
[1]+  Done                    grep main *.c

% grep main *.c > mm &

% ls
```

```
euros*   euros2.c  seno.c   tutor1.c
euros.c  fact2.c   tutor1*  tutor2*
```

E se um processo atrás tentar ler o `stdin` termina com um erro.

3.8 Metacaracteres

Metacaracteres são caracteres que são processados pela `shell` de forma especial:

de redireção `>,<,>>,<<`

encadeamento (`|`)

sequência (`;`)

execução atrás (`&`)

condicionais (`&&,||`)

expansão `*`, `?`, `$`

O significado destes caracteres pode ser ignorado se forem precedidos de `\`

Exemplo 16 *Utilização de Metacaracteres*

```
% echo "lll" \> kk
lll > kk

% echo \$PATH
$PATH

% cat \>
cat: >: No such file or directory

% echo \\
\
```

Uma linha de comando antes de ser executada é dividida em palavras e depois são efectuadas *expansões*, entre elas:

- `~`, é expandido para o nome do caminho absoluto do diretório casa do utilizador
- de variáveis, substitui `$var` pelo seu valor
- de comandos, `` `cmd` ``, executa o comando cmd
- de expressões aritméticas
- de nomes de ficheiros

Se um argumento de um comando estiver entre plicas (`''`) não são efectuadas substituições (expansões) de metacaracteres, variáveis, nomes de ficheiros ou comandos.

Exemplo 17 *Expansões e substituições*

```
% echo '$PWD=`pwd`'
$PWD=`pwd`
% ls '*'
ls: *: No such file or directory
```

Se um argumento estiver entre aspas (`""`) apenas não é feita a expansão de nome de ficheiros (as restantes substituições são efectuadas).

Exemplo 18 *Expansões e a utilização das aspas*

```
% echo "$PATH"
/home/nam/bin:/usr/local/bin:/usr/sbin:/usr/local/bin:
% ls "*"
ls: *: No such file or directory
```

4 Ficheiros de comandos e Controlo de Fluxo

Qualquer sequência de comandos pode ser guardada num ficheiro de texto, e executada invocando o nome do ficheiro (eventualmente com argumentos). São úteis para guardar sequências de comandos habituais e são essenciais para a realização de tarefas de gestão do sistema. Por exemplo:

```
echo "Bem vindo ao $HOSTNAME"
echo -e "Data e Hora:\c" date
echo -e "Voce é: `whoami` \nO seu diretorio corrente é: \c" pwd
echo "Bom trabalho"
```

Um ficheiro de comandos tem de ter permissão de executar:

```
%chmod +x bomdia
% bomdia
Bemvindo ao khayyam
Data e Hora:Wed Nov  7 12:16:38 GMT 2001
Voce é: nam
O seu diretorio corrente é: /home/nam/Aulas/IC/SCRIPTS
Bom trabalho
%
```

Quando se executa um ficheiro de comandos é necessário saber qual a *shell* que se deve invocar. Para tal:

- Se a primeira linha for da forma `#!nomecompleto` o programa executável `nomecompleto` é usado para interpretar o ficheiro. Ex: `#!/bin/bash`, `#!/bin/tclsh`, `#!/bin/wish`.
- Caso contrário, o ficheiro é interpretado pelo comando `sh`.
- Outras linhas inicializadas por `#` são ignoradas pelos interpretadores (servem de comentários)

4.1 Parâmetros

Quando um ficheiro de comandos é executado, são associados alguns parâmetros, que em particular permitem aceder aos argumentos da linha de comandos. Desigam-se por: `$, 0, 1, ..., 9, *, @, #, !, ?`. Os valores deles são:

Parâmetro	Descrição
`$$`	o id do processo da shell
`$0`	o nome do ficheiro de comandos (*script*)
`$1...$9`	cada um dos argumentos do comando
`$*`	todos os argumentos do comando numa única sequência de caracteres
`$@`	a lista de todos os argumentos do comando
`$#`	número de argumentos
`$!`	id do último processo executado atrás
`$?`	estado de saída do último comando executado (0 ou 1)

Não se pode alterar o valore destas variáveis, mas o comando `shift n` permite transladar os parâmetros Pi para P(i-n).

Exemplo 19 *Utilização de parâmetros na linha de comando*

```
% cat esta_ca
who | grep $1

% esta_ca nam
nam      :0       Oct 25 09:41
nam      pts/0    Oct 25 09:41
nam      pts/1    Oct 25 09:41
nam      pts/2    Oct 25 09:41

% cat esta_ca1
who | grep $1| cut -d' ' -f1 |uniq

% esta_ca1 nam
nam

% cat limpa
rm -f a.out *~ core

% cat imprime
lpr $* ; tar cvf guarda$$.tar $*

% imprime ex1.c ex2.c ex3.c
```

4.2 Comando `for`

Para controlo de execução:

Comando	Objectivo	Exemplo

`for` var [`in` valores;] `do` comandos; `done`	Executa a lista de comandos `comandos` uma vez para cada elemento na lista valores. A variável `var` toma o valor de cada um desses elementos. Os `;` podem ser substituídos por mudança de linha.	`for i in aulas/ic/*.apoo aulas/pi/*.c`
		`do`
		`  ls $i`
		`done`

4.3 Comando `case`

Comando	Objectivo	Exemplo

`case` palavra `in`
`pad_1)` comandos_1 `;;`
...
`pad_n)` comandos_n `;; esac`
	Executa a primeira lista de comandos `comandos_i` em que `palavra` é igual ao padrão `pad_i`	`case $op in`
		`  -l) ls -l;;`
		`  -a) ls -a;;`
		`  *) echo "opção errada";;`
		`esac`

4.4 Comandos `if` e `exit`

O comando `if` surge, normalmente, associado ao comando test.

Comando	Objectivo	Exemplo

`if` lista1; `then` lista2; [`else` lista3;] `fi`	Executa a lista de comandos `lista1`. Se o valor de retorno é `0` executa os comandos em `lista2`; caso contrário executa os comandos em `lista3` (se esta existir). Os `;` podem ser substituídos por mudança de linha.	`if test -f prog.c;`
		`  then cat prog.c;`
		`else echo "prog.c não existe";`
		`fi`

`exit`	Terminar a execução do programa	`if ...`
		`then exit`
		`else ...`

4.5 Testes e comparações em *shell*

4.5.1 Código de retorno

Todos os comandos em UNIX retornam um código, denominado por *return status*. Sempre que o comando é executado sem erros, o código retornado é 0. Contudo, se houver alguma falha, o código retornado é diferente de 0.

Exemplo 20 *Código de retorno de comandos*

```
% ls
doc1.txt doc2.txt
% echo $?
0
% rm doc.txt
rm: cannot remove ‘doc.txt’: No such file or directory
% echo $?
1
```

4.5.2 `test` ***expr*** e `[` ***expr*** `]`

Este comando retorna 0 ou 1, dependendo se a avaliação da expressão *expr* retornou sem erros, ou com erros. Permite realizar 3 tipos de testes:

- testes em valores numéricos
- testes em ficheiros
- testes em strings

Valores numéricos (`Num1` op `Num2`):

Operador	Objectivo da operação (comparação)
`-eq`	`Num1` igual a `Num2`
`-ne`	`Num1` diferente de `Num2`
`-gt`	`Num1 > Num2`
`-lt`	`Num1 < Num2`
`-ge`	`Num1 >= Num2`
`-le`	`Num1 <= Num2`

Exemplo 21 *Utilização do comando `test`*

```
% num=5
% test $num -eq 10; echo $?
1
% [ $num -ge 10]; echo $?
1
% [ $num -lt 10]; echo $?
0
% [ $num -ge  "" ]; echo $?
-bash: [: : integer expression expected
2
```

Ficheiros (teste ficheiro):

Teste	Objectivo
`-s`	testa se o ficheiro existe, e se é não-vazio
`-e`	testa se o ficheiro existe
`-f`	testa se é um ficheiro normal (se não é diretório)
`-d`	testa se é diretório
`-r`	testa se tem permissão de leitura
`-w`	testa se tem permissão de escrita
`-x`	testa se tem permissão de execução

Exemplo 22 *Outro tipo de teste*

```
% ls -l
drwxr-xr-x  5 xpto xpto   170 Sep 25 11:12 ic
-rw-r--r--  1 xpto xpto     3 Sep 18 13:50 t1

% [ -d ic ]; echo $?
0
% [ -f t1 ]; echo $?
0
% [ -e t2 ]; echo $?
1
% [ -x t1 ]; echo $?
1
```

Strings

Operador	Objectivo da comparação
`string1 = string2`	testa se `string1` e `string2` são iguais
`string1 != string2`	testa se `string1` e `string2` são diferentes
`string1 \< string2`	testa se `string1` é lexicograficamente inferior à `string2`
`string1 \> string2`	testa se `string1` é lexicograficamente superior à `string2`
Teste	Objectivo
`-z`	testa se o comprimento da string é igual a 0
`-n`	testa se o comprimento da string é diferente de 0

Exemplo 23 *Teste sobre strings (sequências de caracteres)*

```
$ str="lindo dia de sol"
$ [ str = "chuva"]; echo $?
1
$ [ -n str ]; echo $?
0
```

Operadores *booleanos*:

Operador	Objectivo
`-a`	conjunção
`-o`	disjunção
`!`	negação

Para alterar as precedências deve usar `\( \)`.

Exemplo 24 *Composição de testes*

```
% [ -d $HOME -o 4 -ge 3 ]; echo $?
0
% [ ! -d $HOME -o 4 -ge 3 ]; echo $?
0
% [ ! \( -d $HOME -o 4 -ge 3 \) ]; echo $?
1
```

4.5.3 `[[` *expr* `]]`

É uma alternativa mais poderosa ao uso de `test` *expr* ou `[` *expr* `]` para a realização de testes sobre ficheiros ou strings. Permite uma sintaxe mais intuitiva, mais próxima à da linguagem C. Eis alguns dos novos operadores:

Operador	Objectivo
`string1 < string2`	testa se `string1` é lexicograficamente inferior à `string2`
`string1 > string2`	testa se `string1` é lexicograficamente superior à `string2`
`string1==string2`	verifica se o encaixe de padrões entre as strings é possível
`string1=~string2`	verifica se o encaixe de padrões por expressões regulares entre as strings é possível
`&&`	conjunção
`||`	disjunção

Para alterar as precedências pode usar `( )`.

Exemplo 25 *Outro tipo de teste mais poderoso*

```
$ [[ "abc def" == a[abc]*\ ?d* ]]; echo $?
1
$ [[ "abc def" =~ a[abc]*\ ?d* ]]; echo $?
0
$ [[ ! ("abc def" =~ a[abc]*\ ?d* &&  -d $HOME) ]]; echo $?
1
```

4.5.4 `let` ***expr*** e `((` ***expr*** `))`

Permite a avaliação de expressões aritméticas entre inteiros. Face ao resultado da avaliação da expressão aritmética estabelece que o código de retorno é:

- 1, se a expressão é avaliada em 0 (*false*);
- 0, se a expressão é avaliada num valor diferente de 0 (*true*).

Exemplo 26 *Utilização do comando* `let`

```
% (( 12 >= 2 )); echo $?
0
% let x=2 y=2**3 z=y*3; echo $x $y $z
2 8 24
% num=10; let num=num+1; echo $num
11
% let num++; echo $num
12
% num=$((num+1)); echo $num
13
% echo $((num++)); echo $num
13
14
% echo $((++num))
15
```

4.6 Comando `expr`

Operação	Exemplo
Strings	
`match STRING REGEXP`	`expr match ola '.*la'`
`substr STRING POS LENGTH`	`expr substr "bom dia" 4 3`
`index STRING CHARS`	`expr index "bom dia" "ia"`
`length STRING]`	`expr length "bom dia"`

Operação	Exemplo
Aritméticas	
adição + subtração -	`M=‘expr 1 + 1‘`
multiplicação * divisão /	`M=‘expr $M \* 3‘`
módulo %	`M=‘expr $X % 2‘`
Relacionais	
menor \< menor ou igual \<= maior \> maior ou igual \>=	`M=‘expr $X \< 2‘` `if test $M -eq 1 ; then echo TRUE ; fi`
igual == diferente !=	`M=‘expr $X == 2‘` `if test $M -eq 0 ; then echo FALSE ; fi`
Booleanos	
e \& ou |	`M=‘expr $X \< 2 | $X \> 5 ‘` `M=‘expr $X \> 2 \& $X \< 5 ‘`

Nota: deve utilizar o caracter ’para evitar que a shell interprete da forma habitual os caracteres `’*’, ’<’, ’>’, ’(’ e ’)’`.

4.7 Comando `while`

Comando	Objectivo	Exemplo

`while condicao`
`do`
`  comandos`
`done`
	Executa `comandos` enquanto `condicao` for verdadeira (retornar 0)	`while test $n -gt 10`
		`do`
		`  echo $n`
		`  n=‘expr $n - 1‘`
		`done`

4.8 Comando `read`

Comando	Objectivo	Exemplo

`read var`
ou
`read var1 ... varn`
	Lê uma linha do canal *standard* de entrada e coloca-a numa variável. Se for especificada mais do que uma variável, cada palavra da entrada é colocada na variável respectiva.	`read nome`
		`echo Bom dia $nome`

Nota: se houver mais palavras do que variáveis, a última variável fica com todas as palavras que restam.

4.9 Comandos `break` e `continue`

Permitem interromper e continuar para o próximo valor num ciclo

Exemplo 27 *Procurar ficheiros por nome e colocar o seu nome completo num ficheiro, cujo nome é dado como argumento.*

```
case $# in
        1) if test -f $1; then echo "$1 existe"; exit
           else
           while [ 1 -eq 1 ]
           do
             read linha
             case "$linha" in
                  .) echo "Fim"
                  break;;
                  *) find . -name $linha -print >> $1;;
             esac
           done
           fi;;
        *) echo 'modo de usar: $0 arg1' ;;
esac
```

4.10 *Arrays*

Comando	Objectivo	Exemplo

`${arr[*]}`	Devolve todo o array.	`% arr=(aa bb cc dd)`
		`% echo ${arr[*]}`
		`aa bb cc dd`

`${#arr[*]}`	Devolve o comprimento do array.	`% arr=(aa bb cc dd)`
		`% arr[4]=ee`
		`% echo ${#arr[*]}`
		`5`

`${arr[@]}`	Devolve todo o array, elemento a elemento.	`% arr=(aa bb cc dd)`
		`% echo ${arr[@]}`
		`aa bb cc dd`

`${arr[i]}`	Devolve o elemento do array na posiçao `i`.	`% arr=(aa bb cc dd)`
		`% echo ${arr[0]}`
		`aa`

`${arr[@]:i}`	Devolve os elementos do array a partir da posição `i`.	`% arr=(aa bb cc dd)`
		`% echo ${arr[@]:1}`
		`bb cc dd`

`${arr[@]:i:j}`	Devolve os elementos do array entre a posição `i` e a posição `j`	`% arr=(aa bb cc dd)`
		`% echo ${arr[@]:1:2}`
		`bb cc`

4.11 Manipulação de strings

Comando	Objectivo	Exemplo

`${#Z}`	Devolve o comprimento da string `Z`	`% Z="dia de sol"`
		`% echo ${#Z}`
		`10`

`${Z:i:j}`	Devolve a substring de `Z` com início na posição `i` e comprimento `j`.	`% Z="dia de sol"`
		`% echo ${Z:4:2}`
		`de`

`${Z:i}`	Obtém a substring de `Z` com início na posição `i`.	`% Z="dia de sol"`
		`% echo ${Z:4}`
		`de sol`

`${Z#X}`	Remove o menor encaixe da string `X` em `Z`, a partir do início da string `Z`.	`% Z="dia de sol"`
		`% echo ${Z#* }`
		`de sol`

`${Z##X}`	Remove o maior encaixe da string `X` em `Z`, a partir do início da string `Z`.	`% Z="dia de sol"`
		`% echo ${Z##* }`
		`sol`

`${Z%X}`	Remove o menor encaixe da string `X` em `Z`, a partir do fim da string `Z`.	`% Z="dia de sol"`
		`% echo ${Z% *}`
		`dia de`

`${Z%%X}`	Remove o maior encaixe da string `X` em `Z`, a partir do fim da string `Z`.	`% Z="dia de sol"`
		`% echo ${Z%% *}`
		`dia`

`${Z/X/Y}`	Substituir a ocorrência da string `X` em `Z` pela string `Y`.	`% Z="dia de sol"`
		`% echo ${Z/d/D}`
		`Dia de sol`

`${Z//X/Y}`	Substituir todas as ocorrências da string `X` em `Z` pela string `Y`.	`% Z="dia de sol"`
		`% echo ${Z//d/D}`
		`Dia De sol`

5 Expressões Regulares

Uma expressão regular é uma sequência de caracteres e/ou metacaracteres utilizada para representar um conjunto de outras sequências de caracteres. Os metacaracteres têm uma interpretação diferente do seu significado comum.

Caracter	Descrição	Exemplo
`.`	Coincide com qualquer caracter, excepto o `\n`	`".ato"` encaixa com `"gato"`, `"prato"` mas não encaixa com `"atolado"`
`*`	O caracter anterior pode aparecer em qualquer quantidade	`".*ato"` encaixa com `"gato"`, `"prato"`, `"atolado"`
`^`	Coincide com o início da linha	`"^Era"` encaixa com `"Era uma vez ..."`
`$`	Coincide com o fim da linha	`"sempre$"` encaixa com `"...felizes para sempre"`
`[...]`	Coincide com qualquer um dos caracteres listados	`"[gpr]ato"` encaixa com `"gato"`, `"pato"`, `"rato"`, `"prato"`
`[^...]`	Coincide com qualquer um dos caracteres, excepto os listados	`"[^r]ato"` encaixa com `"gato"`, `"pato"`, mas não encaixa com `"rato"`, `"prato"`
`\`	Torna os metacaracteres caracteres comuns.	`"[0-9]\.[0-9]\*[0-9]"` encaixa com `"0.5*2"`
`\<...\>`	Limita uma palavra.	`"\<mente\>"` encaixa com `"mente"` mas não encaixa com `"finalmente"`

5.1 Expressões Regulares Estendidas

Caracter	Descrição	Exemplo
`?`	O caracter anterior pode aparecer ou não.	`"pr?ato"` encaixa com `"pato"`, `"prato"`
`+`	O caracter anterior deve aparecer no mínimo uma vez.	`"goo+gle"` encaixa com `"google"`, `"goooooogle"` mas não encaixa com `"gogle"`
`\{...\}`	O caracter anterior deve aparecer na quantidade indicada.	`"[0-9]\{4\}-[0-9]\{3\}"` encaixa com `"4000-127"` `"[0-9]\{3,\}-[0-9]\{1,3\}"` encaixa com `"4000-127"`, `"400-1"`, `"400-12"`
`(...)`	Faz com que vários caracteres sejam vistos como um só.	`"(calma)?mente"` encaixa com `"mente"`, `"calmamente"`, `"rapidamente"`
`|`	Actua como operador "ou".	`"(calma|rapida)mente"` encaixa com `"calmamente"`, `"rapidamente"`