C-shell - um interpretador de comandos

José João Almeida
José Bernardo Barros
Pedro Rangel Henriques

Fevereiro de 1991

Contents

1 Introdução

Em UNIX, ao contrário do que acontece em outros sistemas operativos, o interface com o utilizador é feito por um programa (**shell**) que não faz parte do sistema operativo.

Desta forma, existem vários interfaces possíveis que correm sobre UNIX, e a mudançca de um interface para o outro é feita muito facilmente.

O algoritmo genérico deste tipo de programas é:

Repetir

Ler linha de comando
Interpretar linha
Executar os comandos necessários

Por defeito, o interface usado com o utilizador é a *Bourne-shell.* Esta shell é um programa bastante compacto mais rápido do que a *C-shell* .

A *C-shell* é um interpretador de comandos com uma sintaxe próxima da do *"C"* que permite:

- abreviar alguns comandos ou sequências de comandos
- guardar uma lista dos comandos executados e poder usá-los na construção de novos comandos.
- criação de novos processos, controlando se estes se fazem com ou sem interacção com o terminal (*foreground* ou *background*).
- escrita de ficheiros de comandos – **scripts** – com estruturas de controlo bastante poderosas.

A *C-shell* pode ser usada de três formas distintas:

1. **Interactivamente**
   Ao introduzir-se uma linha de comando podem ser usadas as seguintes facilidades:
   - uso de abreviaturas
   - uso de partes de comandos anteriores
   - referência a variáveis de ambiente
   - uso de "*wild cards*" na escrita de nomes de ficheiros
2. **Inicialização do ambiente**
   Ao iniciar a sua execução , a *C-shell* executa os comandos que estão no ficheiro **.cshrc** (C-shell *resources*) e eventualmente os comandos que estão no ficheiro **.login** (no caso de se tratar da shell de *login*).

   Desta forma é possível inicializar o ambiente de trabalho, colocando nestes ficheiros os comandos necessários.

3. **Escrita de programas ao nível da shell** Podem-se escrever programas, ditos C-shell *-scripts* que não são mais do que ficheiros de comandos que são interpretados linha a linha.

   Desta forma é possível automatizar certas tarefas poupando esforço e evitando erros.

   As estruturas de controlo são tão poderosas que estes programas podem em muitos casos ser uma boa alternativa à programação numa qualquer outra linguagem, já que não necessitam de qualquer outro suporte (compiladores, linkers, ...)

O texto que se apresenta divide-se em duas partes fundamentais:

Uso interactivo da *C-shell*

Programação em *C-shell*

Apesar de a maioria dos conceitos abordados em cada uma das partes se aplicar também à outra parte, pareceu-nos útil esta divisão.

2 Uso interactivo da *C-shell*

2.1 Reconhecimento da linha de comando

Cada linha de comando é dividida em palavras. O fim de uma palavra é detectado pelo aparecimento de um *tab* ou *espaço*. Exceptuam-se os seguintes casos:

- os caracteres &, |, ;, <, >, (e) formam palavras;
- se os caracteres anteriores forem duplicados (e.g., && ||) também formam palavras;
- o caracter \ seguido de um *newline* é equivalente a um espaço.

Se o introdução da linha de comando não for via teclado (numa *script*), o caracter # significa que daquela posição até ao fim da linha se trata de um comentário.

Todas estas excepções (caracteres especiais) podem ser ultrapassadas se esse caracter for precedido pelo caracter \ .

2.1.1 Comandos

Um comando simples é uma sequência de palavras das quais a primeira é o nome do comando.

O comando formado por comandos simples ou sequências de comandos, separados entre si pelo caracter | chama-se uma *pipeline*.

Numa *pipeline* o *output* de um comando é direccionado para ser o *input* do comando seguinte.

Sequências de *pipeline*'s separadas pelo caracter **;** são executadas sequencialmente.

Se uma *pipeline* ou sequência de *pipeline*'s for englobada por **()** então ela é tratada como um comando simples (que pode por exemplo ser o elemento de uma *pipeline*).

Se uma sequência de *pipeline*'s for terminada pelo caracter **&** então ela é executada em *background*. Neste caso aparece no ecran uma linha da forma
2396
indicando que aquele comando está a ser executado em background pelo processo nº 2396.

2.1.2 Substituições feitas na linha de comando

Antes de executar os comandos, a *C-shell* faz uma determinado número de substituições na linha de comando.

De seguida apresentam-se as substituições que são feitas, pela ordem em que são feitas.

Uso dos comandos anteriores

Como já foi dito a *C-shell* guarda os comandos que se vão executando. O número máximo de comandos que são guardados em simultâneo está armazenado na variável *history* juntamente com o número de ordem desse comando.

A lista dos comandos anteriores pode ser visualizada usando o comando *history*.

Por exemplo, se se fizer:
$ history <CR>↙
poderia aparecer
10 vi manual.tex
11 cat /usr/local/tex/inputs/artigo.sty
12 cd /usr/local/tex/inputs
13 vi artigo.sty
14 cd
15 latex manual
16 pubview manual
17 dvips manual
18 lp manual.ps
19 rm *.ps
20 history

Note -se que, do resultado da execução deste comando, podemos deduzir o valor da variável *history* (= 11).

É possível usar partes de comandos anteriores na construção de novos comandos.

Uma referência a um comando anterior pode ser feita em qualquer ponto da linha de comando e tem a forma: **!comando:palavras:acção** em que:

comando especifica qual o comando a ser usado.
Esta especificação pde ser feita de várias formas:

1. através do número de ordem do comando (e.g. !12 significa o comando nº 12)
2. através de um deslocamento na lista dos comandos; por exemplo !-3 significa o comando 17 (se o actual for o nº 20).
   Como caso particular o comando !-1 (o comando anterior) pode ser abreviado por **!!**).
3. através dos primeiros caracteres do comando; por exemplo **!vi** refere-se ao comando nº 12.
4. **!?aux?** refere-se ao último comando onde aparece a *string* **aux**, i.e., ao comando nº 14.

palavras especifica quais as palavras a serem seleccionadas.
Se for omitida esta parte então toda a linha do comando especificado é seleccionada.
A selecção das palavras faz-se usando uma das formas:

- 0 primeira palavra (nome do comando)
- n **n**-ésimo argumento ((n+1)-ésima palavra)
- | primeiro argumento (equivalente a 1)
- $ último argumento
- x-y desde a posição x até à y.
- -y o mesmo que 0-y.
- x* o mesmo que x-$
- x- desde a posicao x até à ultima (exclusivé).
- * o mesmo que 1-$

O caracter **:** que marca o início da zona de seleção das palavras pode ser omitido desde que esta comece por |, **$**, ***** ou por **$-$**.

acção Se for omitida a acção a executar então não é executada nenhuma acção sobre a parte do comando seleccionada.
As acções possíveis são:

h (head) seleccionar apenas a 1ª parte da palavra;
O comando 12 poderia ter sido feito como **cd !11:1:h**

t (tail) remover a 1ª parte da palavra;
O comando 13 poderia ser feito como **vi *.!11:1:t**

r remover a extensão;
O comando 15 poderia ser feito como **latex !10:1:r**

e seleccionar apenas a extensão;
O comando 19 poderia ser feito como **rm *!18:1:e**

s/*frase*/*subst*/ substituir *frase* por *subst*;
O comando 16 poderia ser feito como **!15:s/latex/pubview/**

% repetir a última substituição ;

g aplicar a acção globalmente, i.e., tantas vezes quantas possível;

p não executar o novo comando, imprimindo-o apenas.

A menos que tenha sido precedida por um **g**, cada acção só é aplicada uma vez.

Se no uso de comandos anteriores não for especificado o comando a seleccionar então é seleccionado o comando anterior; assim, por exemplo !:1 significa o primeiro argumento do comando anterior.

Existe uma forma abreviada de fazer substituições no comando anterior. Em vez de se fazer
$!:s/*frase*/*subst*/
pode-se fazer
$^*frase*^*subst*

Substituição de abreviaturas

A *C-shell* mantem uma lista de abreviaturas que pode ser alterada usando os comandos *alias* e *unalias*.

Depois de cada linha de comando ser lida, ela é decomposta nos vários comandos que a compõe. Para cada um destes testa-se se a primeira palavra (nome do comando) se trata de uma abreviatura. Em caso afirmativo ela é substituida pelo seu significado.

Substituição de variáveis

A *C-shell* tem um conjunto de variáveis que pode ser alterado usando os comandos *set* e *unset*.

O valor de uma variável é uma lista de zero ou mais palavras que podem ser acedidas separadamente.

Existem variáveis que são tratadas como booleanas, interessando apenas se estão activas ou não. É o caso de *verbose* que estando activa significa que todo o *input* será ecoado no ecran.

Uma variável está activa quando é activada (usando o comando **set**) ou quando lhe é atribuido um valor.

Uma variável está inactiva desde que não tenha sido activada ou desde que tenha sido desactivada (usando o comando **unset**).

As substituições de variáveis fazem-se quando na linha de comando aparece:

$name ou **${name}** é substituido pelo valor da variável *name*.

$name**[**selector**] ou **${name[**selector**]**}** é substituido apenas por parte da variável *name*.
O selector pode ser:

- **x-y** significando as palavras desde a x-ésima até à y-ésima.
- **x** a x-ésima palavra.
- **x-** da x-ésima até à última.
- ***** todas as palavras (equivalente a **1-**).

$#name ou **${#name}** é substituido pelo número de palavras de variável *name*.

$$ é substituido pelo número do processo que invocou o actual (pai).

$< é substituido pelo resultado da leitura de uma linha do canal standard de input (*standard input*).
É útil para numa *script* ler alguma coisa do teclado.

Substituição de comandos

Quando um comando é englobado por ` ` então ele é substituido pelo resultado (output) da sua execução .

Substituição de nomes de ficheiros

Numa linha de comando, qualquer palavra que contenha um dos caracteres *****, **?**, [, **{** é considerada como um padrão e é substituida por uma lista de nomes de ficheiros que verifica tal padrão.

As substituições que são feitas são as seguintes:

1. o caracter ***** significa uma qualquer sequência de caracteres.
2. o caracter **?** significa um qualquer caracter.
3. uma sequência de caracteres englobados por [ ] significa um qualquer caracter da sequência
4. uma sequência de strings separadas por **,** e englobada por **{ }** significa qualquer uma das sequências.
5. o caracter ~ no inicio da palavra significa a directoria de login (cf. com $HOME)

Um facto importante é que, excepto quando é usada a forma **{ }**, a lista de palavras que satisfaz o padrão é ordenada por ordem alfabética.

Assim, enquanto que *[cba]d* é substituido por **ad bd cd**, *{a,c,b}d* é expandido para **ad cd bd**.

Algumas destas substituições podem ser evitadas se forem usadas **"** ou **´**.

Uso de " Evita as substituições dos nomes de ficheiros.

Uso de ´ Evita as substituições dos nomes dos ficheiros e das variáveis.

Exemplo Considere-se que existem definidos:

abreviaturas
dir ls -la
print pr \!* | lp

comandos anteriores
1 vi fich1.c
2 cc -c fich1.c
3 mv fich1.c prog1.c
4 ls -l prog

variáveis
Trab prog
home /usr/acct/util

ficheiros na directoria corrente
prog1.c fich1.o manual manual.tex a.out prog

ficheiros na directoria *prog*
carta.c hello.c talk.pas matriz.f

Vejamos as modificações que se produzem quando se introduzem as seguintes linhas de comando:

1. **print $home/!4:2/*.c**

   (a) **!4:2** é substituido por **prog**

   (b) **print** é substituido por **pr ... | lp**

   (c) $home é substituido pelo valor da variável **home**

   (d) ***.c** é substituido por **carta.c hello.c**

   É executado o comando **pr /usr/acct/util/prog/carta.c /usr/acct/util/prog/hello.c | lp**

2. **print "$home/!4:2/*.c"**

   (a) **!4:2** é substituido por **prog**
   (b) **print** é substituido por **pr ...** | **lp**
   (c) $home é substituido pelo valor da variável **home**

   É executado o comando **print /usr/acct/util/prog/*.c | lp**

3. **print ´$home/!4:2/*.c´**

   (a) **!4:2** é substituido por **prog**
   (b) **print** é substituido por **pr ...** | **lp**

   É executado o comando **print $home/prog/*.c | lp**

2.2 Redireccionamento

Em UNIX, existem pre-definidos três ficheiros especiais:

stdin Canal standard de entrada de dados.
É deste ficheiro que se lê quando se faz um *scanf* em "C" ou um *read* em "Pascal" sem especificar o nome do ficheiro.

stdout canal standard de saida de dados.
É neste ficheiro que se escreve quando se faz um *fprintf* em "C" ou um *write* em "Pascal" sem especificar o nome do ficheiro.

stderr canal standard de diagnóstico.
É neste ficheiro que, por exemplo, se escrevem os erros de uma compilação .

Por defeito tanto *stdout* como *stderr* encontram-se associados ao ecran, enquanto que *stdin* se encontra associado ao teclado.

Tanto o canal standard de entrada de dados *(standard input)* como o canal de saída de dados (*standard output*) podem ser redireccionados para outro ficheiro.

2.2.1 Do *standard input*

< **name** Abre o ficheiro *name* e este passa a ser o *standard input*

<< **palavra** Lê o *standard input* até que apareça uma linha igual a *palavra* terminando aí a leitura.

2.2.2 Do *standard output*

$>$ **name** O ficheiro *name* é apagado (caso exista) e depois usado como *standard output*.

$>>$ **name** Identico ao anterior só que o ficheiro não é apagado e o *output* é acrescentado a *name*.

2.2.3 Do *standard error*

Não existem formas directas de redireccionar o canal standard de diagnóstico (*stderr*) para um ficheiro. Existem contudo formas de redireccionar tanto este como o *standard output* para um mesmo ficheiro.

$>$**& name** Tanto o *standard output* como o canal standard de diagnóstico (*standard error*) são redireccionados para *name*. O conteudo anterior de *name* é apagado.

$>>$**& name** Identico ao anterior só que o ficheiro não é apagado e as escritas são feitas no fim do ficheiro (append).

Uma forma de redireccionar apenas o canal de diagnóstico para um ficheiro é fazendo **(comando $>$ console) $>$& erro**

2.3 *Pipe*'s

Ao juntarmos dois comandos usando o símbolo |, o canal de entrada do segundo comando passa a estar ligado ao canal de saída do primeiro comando.

Existe contudo uma diferença entre fazer **com1** | **com2** e **com1 $>$ temp ; com2 $<$ temp ; rm temp**.

A diferença reside não no uso de um ficheiro intermédio **temp**, mas na sincronização dos comandos **com1** e **com2**. Assim, enquanto que no 2º caso **com2** só começa a ser executado depois de **com1** ter terminado, no 1º caso os comandos são executados concorrentemente.

Vejamos um caso concreto. Seja **com1** um programa que escreve números inteiros até que seja interrompido (ciclo infinito). Seja **com2** o comando *head* que escreve as 10 primeiras linhas de um ficheiro.

Ao fazermos **com1** | **com2** o que acontece é que depois de **com1** produzir 10 linhas de texto, **com2** termina a sua execução e envia um sinal de interrupção a **com1**. Se usássemos a outra forma, o comando não terminaria.

Uma outra variante de *pipe* é o uso do conectivo |& que faz com que o canal de entrada do 2º comando seja ligado tanto ao canal de saida do 1º comando como ao seu canal de diagnóstico.

3 Programação em *C-shell*

3.1 Estruturas de controlo

3.1.1 Condicionais

if (*expr***)** *comando*

Se o resultado de *expr* for verdadeiro então executa-se *comando*. *Comando* tem de ser um comando simples, i.e., não pode ser uma sequência de comandos (separados por ; ou por |).

if (*expr1***) then**
...
else if (*expr2***) then**
...
else
...
endif

Idêntico ao anterior. Aqui os comandos podem não ser simples. Qualquer número (mesmo 0) de pares **else if (***exp***) comandos** é permitido.
As palavras **else** e **endif** devem aparecer no inicio de linhas.

switch (*string***)**
case *str1***:**
...
breaksw
...
default:
...
breaksw
endsw

Tem uma semântica semelhante ao *statement switch* do *"C"*.
Nas labels, os caracteres *****, **?** e **[...]** têm o mesmo significado que foi referido nos nomes dos ficheiros.
as expressões **switch (***string***)**, **case** *str* e **endsw** devem aparecer no inicio das linhas.

3.1.2 Repetitivas

repeat *n comando*

Comando é executado *n* vezes.
Também aqui *comando* deve ser um comando simples.

while (*expr*)
...
end

Semântica semelhante á do *"C"*.
As palavras **while** e **end** devem aparecer no inicio de uma linha.

foreach *name* (*lista*)
...
end

À variável *name* é-lhe sucessivamente atribuido um elemento de *lista* e executados os comandos. As palavras **foreach** e **end** devem aparecer no inicio de uma linha.

O fluxo normal descrito para os comandos **while** e **foreach** pode ser alterado se forem usados os comandos:

break

Faz com que o ciclo termine. No entanto os restantes comandos dessa linha ainda são executados.

continue

Volta-se ao primeiro comando do ciclo, depois de executar os comandos desta linha.

3.1.3 Incondicionais

source *nome* faz com que sejam executados os comandos que se encontram no ficheiro *nome*. É de alguma forma semelhante à invocação de um procedimento.
Estas chamadas podem ser aninhadas.

exec *comando* *Comando* é executado em vez da script actual.
A diferença entre este comando e o anterior é que enquanto que no anterior, depois de executados os comandos de *nome* o controlo volta à *script* actual, neste comando tal não acontece; quando *comando* terminar a sua execução a *script* termina.

exit(expr) a script acaba a sua execução e retorna o valor de *expr*.
Se esta expressão for omitida, o valor que é retornado é o valor da variável *status*.

goto *palavra* É procurada uma linha que contenha **palavra:** e continua a execução na linha seguinte a esta.

onintr Serve para controlar a acção a tomar aquando de uma interrupção (por exemplo ˆC).
Pode ser usada de três maneiras diferentes:

- **onintr** faz com que a acção tomada seja a que é definida por defeito, i.e., terminar a execução da script.
- **onintr -** faz com que todas as interrupções sejam ignoradas.
- **onintr label** faz com que após uma interrupção se faça **goto** *label*.

3.2 Outros Comandos

alias Serve para visualizar ou alterar as abreviaturas. Pode ser usado de três maneiras diferentes:

- **alias** Mostra todas as abreviaturas que estão definidas.
- **alias** *name* Mostra o significado da abreviatura*name*.
- **alias** *name lista de pal* Faz com que *name* passe a ser uma abreviatura de *lista de pal*.
  Note-se que *lista de pal* pode ser um comando não simples, por exemplo, poderíamos definir
  alias print ´ pr \!* | lp´ para imprimir um determinado ficheiro depois de o passar pelo filtro *pr*.

Uma abreviatura muito apreciada pelos utilizadores do DOS é:
alias cd ´cd \!* ; set prompt=$pwd \ \!\>´

unalias *padrão* Todos as abreviaturas que verificam *padrão* são removidas.

chdir *name* ou **cd** *name* Muda a directoria corrente para *name*.
Se *name* for omitido, muda para a directoria **$home**.

echo -n *lista de palavras* escreve no ecran em lista de palavras. Se for chamado com a opção **-n** não escreve um <CR>↙no fim.

glob *lista de palavras* Semelhante ao anterior só que cada palavra de *lista de palavras* é terminada com um caracter nulo.

history Como já foi mencionado mostra os últimos comandos executados. O número máximo de comandos que é guardado está na variável *history*.

kill -sig pid Envia o sinal *sig* aos processos *pid*. Os sinais que podem ser enviados são obtidos fazendo **kill -l**.

login ou **logout** Termina a presente execução da shell de *login*.
Embora diferentes este dois comandos têm o mesmo efeito. De facto **login** substitui a presente *shell* por uma shell de *login*, enquanto que **logout** termina a presente shell substituindo-a por uma *shell* de *login*.
A existência destes dois comandos liga-se ao facto de manter compatibilidade com outras *shell*'s

nice *+prioridade comando* Serve para modificar a prioridade de execução de um comando.
Se *+prioridade* for omitido então ela é assumida como **4**. Se *comando* for omitido então a prioridade especificada é aplicada aos restantes comandos (da shell ou da script).

nohup *comando* faz com que o processo que vai executar *comando* não seja interrompido quando o processo *pai* terminar (i.e., quando receber um sinal de *hang-up*).
Pode ser ainda usado em scripts sem especificar *comando*, significando que todos os restantes comandos da script não serão interrompidos nessas condições .

rehash
A *C-shell* mantem uma tabela de *hash* com todos os nomes dos comandos utilizáveis (para poupar tempo na procura). Esta tabela é inicializada na entrada do sistema (login), donde se acrescentarmos um comando aos comandos disponíveis esse comando não é encontrado.
O comando **rehash** recalcula esta tabela de *hash*.

unhash Faz com que deixe de se usar a referida tabela de *hash*.

set Alterar ou visualizar o valor de variáveis.
Este comando pode ser usado de várias maneiras:

- **set** Mostra o valor de todas as variáveis.
- **set** *name* Faz com que a variável *name* fique activa e o seu conteudo seja uma lista vazia de palavras.
- **set** *name=palavra* Coloca na variável *name* o valor *palavra*.
- **set** *name[n]=palavra* Coloca *Palavra* na *n*-ésima componente da variável *name*
- **set** *name=(lista de palavras)* Atribui *lista de palavras* à variável *name*.

unset *padrão* Todas as variáveis cujo nome verifica *padrão* são removidas.

setenv *name palavra* Coloca na variável de ambiente *name* o valor *palavra*.

shift *name* As componentes da variável *name* são deslocadas uma posição para a esquerda, perdendo por isso a primeira. (É de alguma forma semelhante a fazer *name* $\leftarrow$ **tail** (*name*)).
Se não for especificado *name* então a variável que é deslocada é **argv**, i.e., a lista de parâmetros com que a script (ou shell) foi invocada.

time *comando* Executa *comando* e diz quanto tempo é que a sua execução demorou.
Se *comando* for omitido, dá o tempo que a presente script (ou shell) já gastou.

umask *valor*
Em Unix, associado a cada ficheiro, existe uma palavra de 9 bits $< b_0 b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 >$ que representa as permissões de acesso a esse ficheiro. Estas premissões distinguem três grupos de utilizadores: **owner** refer-se ao utilizador que criou o ficheiro; **group** refere-se a todos os utilizadores que pertençam ao mesmo grupo do *owner*; **others** refer-se a todos os outros utilizadores. O significado desses bits é o seguinte:

$\mathbf{b}_0$ Leitura para o dono

$\mathbf{b}_1$ Alteração para o dono

$\mathbf{b}_2$ Execução para o dono

$\mathbf{b}_3$ Leitura para os elementos do mesmo grupo

$\mathbf{b}_4$ Alteração para os elementos do mesmo grupo

$\mathbf{b}_5$ Execução para os elementos do mesmo grupo

$\mathbf{b}_6$ Leitura para todos os outros elementos

$\mathbf{b}_7$ Alteração para todos os outros elementos

$\mathbf{b}_8$ Execução para todos os outros elementos

O comando **umask** *valor*, em que *valor* é o número octal que representa o complemento para **1** da palavra de acesso, fixa como *valor* os modos de acesso com que os ficheiros são criados a partir desta altura.
Assim, após **umask 000** todos os ficheiros criados têm todos os acessos, enquanto que após **umask 023**, todos os ficheiros passam a ter todos os acessos para o dono (owner), acesso de leitura e execução para o grupo e acesso de execução para os outros (others).
Se for omitido *valor* então mostra os modos de acesso com que os ficheiros serão criados.

wait Faz com que a presente shell espere por que todos os processos em *background* terminem.

@ É usado para calcular valores de variáveis. Pode ser usado das seguintes formas:

- **@** Escreve o valor de todas as variáveis da shell.
- **@ name = expr** calcula o valor de *expr* e coloca-o na variável *name*.
- **@ name[index] = expr** calcula o valor de *expr* e coloca-o na *n*-ésima posição da variável *name*.

Se *expr* for uma expressão relacional (que use os operadores >, <, = ou |) então deve ser englobada por **(...)**.

Em *expr* podem aparecer a maioria dos operadores *"C"*. Assim é possível escrever **var++** significando que *var* deve ser incrementada.

3.3 Variáveis Pré-definidas e de Ambiente

As variáveis que estão disponíveis podem-se dividir em dois grupos:

1. variáveis de ambiente - que são alteradas usando os comandos **setenv** e **unsetenv**. Estas variáveis são globais às scripts e qualquer modificação é visível após a *script* terminar.

2. variáveis locais - que são modificadas usando os comandos **set** e **unset**. Estas variáveis são locais à script onde são declaradas. Se uma *script* alterar o valor de uma desta variáveis, essa modificação só produz efeito fora da *script* se ela tiver sido definida fora da *script*.

Note-se o paralelismo entre as estas definições e as definições de variáveis locais e globais a procedimentos.

As expressões seguintes só fazem sentido quando invocadas de uma *script*:

$0 é substituido pelo nome do ficheiro de onde o comando está a ser lido.

$n ou **${n}** é substituido pelo n-ésimo parâmetro com que a *script* foi invocada.

$* é substituido pela lista de parâmetros com que a script foi invocada.

$?name ou **${?name}** é substituido por "1" se a variável *name* estiver activa e por "0" no caso contrário.

$?0 é substituido por "1" se for conhecido o nome do ficheiro donde estão a ser lidos os comandos e por "0" no caso contrário.

Neste capítulo faz-se uma descrição das variáveis que têm um significado especial para a *C-shell* . Destas, as variáveis **argv**, **home**, **path**, **prompt** e **shell** são inicializadas pela *C-shell* .

A *C-shell* copia o valor das variáveis de ambiente USER, TERM, HOME para as variáveis **user term** e **home** e qualquer alteração a estas últimas é feita às primeiras.

argv Contém os argumentos da shell (ou da script).

cdpath Contém uma lista das directorias (diferentes das subdirectorias da actual) para as quais se pode fazer **cd** directamente.

echo Está activa quando a shell é invocada com a opção **-x**.
Faz com que cada comando seja ecoado antes de ser executado.

history Contém o número de comandos anteriores que é guardado.

home Contém a directoria base do invocador da shell. Esta informação é retirada do ficheiro */etc/passwd*.

Sempre que num nome dum ficheiro aparece o caracter ˜ ele é substituido pelo valor desta variável.

ignoreeof Se estiver activa, a shell ignora o caracter EOF (ˆD), como caracter de fim de entrada de comandos. Deste modo, para terminar a execução da shell é necessário fazer **logout**.

mail Contém os ficheiros (directorias) onde a shell procura o correio do utilizador.

noclobber Se esta variável estiver activa, verificam-se as seguintes restrições no redireccionamento do output:

- *comando* > *fich* só é possível se o ficheiro *fich* não existir.
  Para ultrapassar esta imposição deve-se fazer *comando* >! *fich*.
- *comando* >> *fich* só é possível se o ficheiro *fich* existir.
  Para ultrapassar esta imposição deve-se fazer *comando* >>! *fich*.

noglob Se estiver activa não se fazem as expansões dos nomes de ficheiros (tal como referidas atrás).

nonomatch Se estiver activa, não é considerado um erro se nenhum nome de ficheiro verificar os padrões dados.

path Cada palavra desta variável significa uma directoria onde se devem procurar os comandos.
Se não contiver nenhuma palavra, apenas se procuram comandos na directoria actual.
Qualquer alteração a esta variável implica normalmente o uso do comando *rehash*, para actualizar a tabela de *hash* dos comandos.

prompt Contém a string que aparece antes da entrada interactiva de um comando.

shell Contém o nome do ficheiro onde reside a shell.

status Contém o estado devolvido pela execução do último comando.
Se esse comando termina *anormalmente* então à variável **status** é adicionado **0200**.
Normalmente os comandos retornam **1** em caso de insucesso e **0** em caso de sucesso.

time Se estiver activo e tiver o valor **n** então após qualquer comando que demore mais do que **n** segundos a executar é escrita uma linha a informar o tempo que esse comando demorou.

verbose Se estiver activa (o que acontece por exemplo, se a shell for invocada com a opção **-v**) faz com que todos os comandos sejam ecoados (após a substituição de abreviaturas).

4 Alguns Exemplos de *scripts*

4.1 Uso de Estruturas de controlo

Nesta secção apresentam-se exemplos de *scripts* que, embora com um interesse muito pequeno, demonstram o funcionamento de alguns comandos específicos.

O exemplo escolhido foi o de fazer uma script que escrevesse a data em português.

Acerca deste exemplo convém referir:

- A primeira linha **#!/usr/ucb/csh** significa que esta script se destina a ser executada pelo programa */usr/ucb/csh* (a *C-shell*).
- O comando **date** dá como resultado uma linha do tipo

Mon Jan 28 16:48:56 EST 1991.

4.1.1 Condicionais

```
#!/usr/ucb/csh
set d=`date`
switch ($d[1])
case "Sun":
set dia="Domingo" ; breaksw
case "Mon":
set dia="Segunda" ; breaksw
case "Tue":
set dia="Terca" ; breaksw
case "Wed":
set dia="Quarta" ; breaksw
case "Thu":
```

```
set dia="Quinta" ; breaksw
case "Fri":
set dia="Sexta" ; breaksw
case "Sat":
set dia="Sabado" ; breaksw
endsw
switch ($d[2])
case "Jan":
set mes="Janeiro" ; breaksw
case "Feb":
set mes="Fevereiro" ; breaksw
case "Mar":
set mes="Marco" ; breaksw
case "Apr":
set mes="Abril" ; breaksw
case "May":
set mes="Maio" ; breaksw
case "Jun":
set mes="Junho" ; breaksw
case "Jul":
set mes="Julho" ; breaksw
case "Aug":
set mes="Agosto" ; breaksw
case "Sep":
set mes="Setembro" ; breaksw
case "Oct":
set mes="Outubro" ; breaksw
case "Nov":
set mes="Novembro" ; breaksw
case "Dec":
set mes="Dezembro" ; breaksw
endsw
echo "$dia, $d[3] de $mes de $d[6]"
```

4.1.2 Repetitivas

foreach

```
#!/usr/ucb/csh
set d=`date`
set diasi=(Mon Tue Wed Thu Fri Sat Sun )
set mesp=(Janeiro Fevereiro Marco Abril Maio Junho Julho Setembro \
          Outubro Novembro Dezembro)
```

```
set diasp=(Segunda-feira Terca-feira Quarta-feira Quinta-feira \
           Sexta-feira Sabado Domingo )
set mesi=(Jan Feb Mar Apr May Jun Jul Aug Sep Nov Dec)
foreach dia ( $diasi )
if ($dia == $d[1]) then
   break
else
   shift diasp
endif
end
foreach mes ($mesi)
if ($mes == $d[2]) then
   break
else
   shift mesp
endif
end
echo "$diasp[1], $d[3] de $mesp[1] de $d[6]"
```

while

```
#!/usr/ucb/csh
set d=`date`
set diasi=(Mon Tue Wed Thu Fri Sat Sun )
set mesp=(Janeiro Fevereiro Marco Abril Maio Junho Julho Setembro \
          Outubro Novembro Dezembro)
set diasp=(Segunda-feira Terca-feira Quarta-feira Quinta-feira \
           Sexta-feira Sabado Domingo )
set mesi=(Jan Feb Mar Apr May Jun Jul Aug Sep Nov Dec)
while ($diasi[1] != $d[1])
   shift diasp
   shift diasi
end
while ($mesi[1] != $d[2])
   shift mesp
   shift mesi
end
echo "$diasp[1], $d[3] de $mesp[1] de $d[6]"
```

4.2 *Scripts* Especiais

Nesta secção apresentam-se alguns exemplos típicos de ficheiros **.login**, **.logout** e **.cshrc**

4.2.1 *.login*

```
echo -n "Ultima entrada: "; cat ~/.ulogin
echo -n "Ultimo logout: "; cat ~/.ulogout
echo ""
echo ""
echo -n "Data :"; date | tee ~/.ulogin
echo ""
echo "Utilizadores"
who
if (-e $mail/$user) then
   echo "CORREIO"
   mail
else
   echo "NAO HA CORREIO"
endif
```

4.2.2 *.cshrc*

```
set history=20
set path=(. /bin /usr/bin /usr/tex/bin )
alias a alias
a h history
a type cat
a ren mv
a meu "chmod 755"
a del rm
a dir ls -l
a cd 'cd \!* ; set prompt=`pwd`\ \!\>'
set ignoreeof
limit  coredumpsize 0
```

4.2.3 *.logout*

```
date | tee ~/.ulogout
```

4.3 Outros Exemplos

4.3.1 Procura

A *script* que que se apresenta pressupõe que existe (na directoria corrente) o ficheiro *projectoX* onde se encontram os nomes dos ficheiros relacionados com um determinado projecto.

O objectivo desta script é o de fazer um *"cross-reference"* das palavras que entram como parâmetro da *script* nos ficheiros que são mencionados em *projectoX*.

```
#!/usr/ucb/csh
if (#$argv == 0) then               # foi invocado sem parametros
   echo "$0 deve ser invocado com pelo menos um parametro"
   exit 1                           # saida com insucesso
endif
set lista = `cat projectoX`         # lista dos ficheiros
foreach palavra $argv[*]
    grep -n palavra $lista
    echo ""
end
```

4.3.2 Alteração

Na *script* que a seguir se apresenta, parte-se também do princípio que existe o ficheiro *projectoX*.

Aqui pretende-se alterar todas as ocorrências de uma string por outra.

```
#!/usr/ucb/csh
if ($#argv != 2) then                   # Sintaxe incorrecta
     echo "Sintaxe : $0 pal1 pal2"
     exit 1                             # saida com insucesso
endif
foreach fich (`cat projectoX`)
    mv $fich $fich.bak
    sed s/$1/$2/g $fich.bak > $fich
end
echo -n "Pretende apagar as versoes anteriores? (S/N)"
switch ($<)
case [Ss] :
foreach fich (`cat projectoX`)
    rm $fich.bak
end
breaksw
default :
echo 'Versoes anteriores guardadas com a extensao ".bak" '
endsw
```

4.3.3 mdosget

Muitos dos utilizadores de Xenix já se depararam com o problema de copiar de uma disquette vários ficheiros. Existe para tal fim o comando **dosget** que tem como grande inconveniente não permitir o uso de "*wild-cards*" no nome do ficheiro.

A script que a seguir se apresenta tenta ultrapassar tal problema.

```
#!/usr/ucb/csh
if ($#argv == 0) then
   echo "Sintaxe : $0 drive:ficheiro"
   exit 1
endif
foreach f (`dosls $1`)
   dosget $f
end
```

4.3.4 Menu

A *script* seguinte pretende ser uma *shell* que só permite que o utilizador execute determinados programas.

Parte-se do princípio da existência do ficheiro **.ecran** na directoria **$HOME** que contém o texto do menu a apresentar ao utilizador.

Este ficheiro seria:

```
                     ==================
                    |  Menu de opcoes  |
   __________________==================________________________
  /____________________________________________________________\
 //                                                            \\
||    E  ------------ Editar um ficheiro                        ||
||    C  ------------ Compilar um programa "C"                  ||
||    M  ------------ Ler o correio                             ||
||    D  ------------ Listar os nomes dos ficheiros existentes  ||
||    F  ------------ Terminar a sessao                         ||
||                                                              ||
 \\____________________________________________________________//
  \____________________________________________________________/
```

O texto da *script* seria:

```
#!/usr/ucb/csh
onintr fim
while (1)
cat ~/.ecran
echo -n "Opcao : "
switch ($<)
case [Ee]:
   echo -n "Nome do ficheiro :"
```

```
   vi $<
   breaksw
case [Cc]:
   echo -n "Nome do ficheiro :"
   cc $<
   breaksw
case [Mm]:
   mail
   breaksw
case [Dd]
   ls -l
   breaksw
case [Ff]:
   goto fim
   breaksw
default:
   echo "Opcao desconhecida"
   breaksw
endsw
end
fim:
logout
```