Apontamentos de apoio às aulas de UNIX

Programação III

Simão Melo de Sousa e Paul Andrew Crocker
Departamento de Matemática/Informática da UBI

19 de Fevereiro de 2001

Conteúdo

Aviso Prévio

A redação deste documento baseou-se fortamente na bibliografia indicada. Portanto a leitura e aprendizagem directa pelas obras originais é altamente recomendada, e mesmo essencial à compreensão profunda das noções aqui apresentadas.

1 Introdução

- Linguagem C, sistema Unix : "Gémeos superdotados"que nasceram em 1970, mas que se tornaram agora independentes.
- A nossa abordagem:
  1. Nada mais de que uma utilização prática do sistema Unix.
  2. Estudo da programação em linguagem C.
  3. Uso de ferramentas de desenvolvimento Unix e C.
- Vantagens do sistema Unix:
  1. Conjunto completo de ferramentas de desenvolvimento como editores, compiladores, suporte completo e variado para comunicações
  2. Sistema multi-tarefa e multi-utilizador com interface gráfica (o X-windows).
  3. Sistema completamente parametrizável, com comandos simples e extensíveis (por exemplo a composição de comandos).
  4. Linguagem de comandos completos e potentes ($\Rightarrow$programação).
  5. Sistema escrito em C $\Rightarrow$ portabilidade, eficiência.
  6. A filosofia do sistema faz com que seja potente, espantosamente rico, aberto, elegante (ao nível da resolução de problemas), e que só tem por limites a imaginação e criatividade do utilizador.

7. A transparência e as boas propriedades da sua estrutura (por exemplo dos *"systems calls"*) permite a escrita de aplicações suficientemente independentes em relação ao suporte físico, o que permite com que o programador não se preocupa demasiado com as características do material.
8. A estrutura dos ficheiros Unix facilita a escrita de aplicações (qualquer ficheiro = fluxo não ordenado de bytes).

- Desvantagens do sistema Unix:

1. Várias versões (diferenças nos "system calls"), mas a norma POSIX (1990) vai resolvendo esses problemas
2. O sistema Unix é um Ferrari que tem de ser conduzido por um mecânico, por oposição aos carros sem carta que toda gente pode usar sem grande esforço (o Win95-98) :).
3. Processadores de texto potente mas difíceis de dominar (Por exemplo o LaTeX e o Emacs), pelo menos, não tão fáceis de usar como o MsWord. Se formos má lingua, podemos extender este ponto ao sistema Unix em geral, mas, mais uma vez, quem disse que era fácil conduzir um Ferrari? :) [1]
4. O "user interface"ainda não é suficientemente "user friendly"(os numerosos comandos embora potentes são constituidos por muitas opções e memorizá-las requer outra memória outra além da do computador :).

- Vantagens da linguagem C:

1. Linguagem madura, e padronizada (norma ANSI, 1988). Por isso portável. Além de que a biblioteca "standard"fornece um conjunto mais que suficiente de primitivas de programação.
2. Utiliza de forma extensiva todos os recursos do material (e.g. os apontadores).
3. Grande número de bibliotecas de primitivas que fortalecem o "kernel"do C que é afinal bastante simple e pequeno.
4. "Assembly"de alto nível.

- A linguagem C, os inconvenientes: Essencialmente os defeitos das suas qualidades,

1. Falta de uma tipagem rigorosa (que pode conduzir a erros graves de programação e a *b*ugs, se houver descuido pela parte do programador).
2. A riqueza da linguagen e as facilidades sintácticas podem conduzir a um programa eficiente mas incompreensível.

Começamos, então, por introduzir o sistema operativo Unix na óptica do utilizador.

[1] O Linux foi o terreno de algumas batalhas estes últimos meses, batalhas travadas pelas grandes companhias de software e hardware. O resultado foi a disponibilização de algum software "Freeware"como o WordPerfect, que é um dos concorrentes ao MsWord, como também do aparecimento de alguns pacotes Freeware "Office"e companhia, por exemplo, o StartOffice.

2 O Que é um Sistema Operativo?

<table>
<tr><td colspan="2">Aplicações</td></tr>
<tr><td>Shell</td><td>Compiladores, Editores</td></tr>
<tr><td colspan="2">Chamadas ao sistema</td></tr>
<tr><td colspan="2">Linguagem máquina</td></tr>
<tr><td colspan="2">Micro programas</td></tr>
<tr><td colspan="2">Dispositivos físicos</td></tr>
</table>

1. Os três últimos níveis são os níveis relacionados com o "hardware"e com a camada de "software"a que ele está directamente ligado:

   (a) Dispositivos físicos O Hardware:

   - um ou mais processadores
   - memória central
   - relógio interno
   - periféricos de entrada/saída como os discos, teclados, impressoras, scanners, ecrãs...

   (b) Os programas de contrôlo directo que envolvem micro programas e a linguagem máquina:

   - Micro-programa: software localizado na "ROM": é um interpretador capaz de executar as instruções da linguagem máquina.
   - Linguagem máquina: conjunto relativamente pequeno de primitivas dedicado para mover valores "atómicos", registos, efectuar cálculos e comparações de esses valores atómicos.

2. O nível mais alto é o conjunto das aplicações que resolvem os problemas do utilizador, como a gestão de bases de dados, os processadores de texto, jogos, etc...

   Se cada uma dessas aplicações devia tomar em conta directamente da gestão dos recursos físicos, o trabalho do programador seria demasiado "titanesco", ou seja quase irrealizável. Este problema é resolvido colocando entre as aplicações e o material uma camada de software que realiza tarefas globais e oculta a complexidade do material. É a este "intérprete"do hardware ou intermediário, que designamos por sistema operativo.

3. É preciso distinguir no sistema operativo os vários níveis que o constituem :

   (a) Os System Calls: são funções (cujo código está em memória) que podem ser usados directamente pelos programas e que estão ligados a acções sobre o material (e.g. o "open"para abrir um ficheiro).

   (b) O Shell, uma caixa de ferramentas do sistema, de alto nível: Funções sobre o sistema de ficheiros, sobre a comunicação, arquivagem, impressão, etc...

   Um shell possuí também uma linguagem de comandos permitindo extender, fazer a composição ou programar funções sistemas de alto nível: os **shell scripts**.

   (c) Ferramentas de desenvolvimento: em Unix, são fornecidas em "standard"numerosas ferramentas tais como compiladores, "debuggers", editores de "links", meta-compiladores (lex e Yacc) , etc...

3 Sessão Unix

3.1 Avisos Prévios:

1. Unix: sistema operativo que permite (ou que é baseado em) políticas de restrições para um uso optimal e seguro dos recursos físicos e lógicos.

   $\Longrightarrow$ Só um utilizador devidamente autorizado pode trabalhar num sistema Unix.

2. Os sistemas Unix são baseado na linguagem C.

   $\Longrightarrow$ Distinção entre maiusculas e minusculas.

3. Serão descritos, nas secções seguintes, comandos nas suas formas mais usuais. Cada comando disponibiliza numerosas e importantes opções que escolhemos, por razões práticas, não sempre descrever. Recomenda-se então ao leitor completar os seus conhecimentos pela consulta dos manuais ou da ajuda on-line do sistema Unix.

3.2 Abertura de uma Sessão Unix

O sistema Unix é um sistema multi-utilizador, por isso é necessário que ele saiba reconhecer o utilizador para poder atribuir lhe o ambiente de trabalho que lhe é próprio. O sistema Unix espera a conexão de um novo utilizador por uma mensagem do género

```
Digital UNIX (ciunix.ubi.pt) (ttyp1)

login:
```

Em resposta a esta mensagem o utilizador introduz o username (ou login name) pelo qual o sistema o conhece. A mensagem seguinte aparece então:

```
password:
```

a qual o utilizador reponde pela senha associado ao username. Se a conecção for recusada aparecerá então a mensagem seguinte:

```
login incorrect
```

O êxito ou o insucesso de uma conexão define-se, ao nível dos protocolos de conexão, pela consulta e comparação com os dados presentes no ficheiro `/etc/passwd` que contem a lista dos utilizadores do sistema, as senhas encriptadas de cada um deles, assim como informação adicional como o shell de aranque, o "user id", o nome do utilizador, o "group id", assim como a localização do directório pessoal de cada utilizador.

No caso de uma conexão ser autorizada, o sistema iniciará todo um processos de inicialização de ambiente de trabalho, isto é

- atribuição de um shell ao utilizador (definido no `/etc/passwd`)
- Leitura dos ficheiros de configuração próprios do sistema.
- Leitura dos ficheiros de configuração próprios do utilizador (definidos pelo administrador, ou simplesmente pelo utilizador)

3.3 Desconexão

Basta mandar ao shell principal o caracter **eof**, cujo valor é habitualmente `ctrl-d`. Outra forma é o envio do comando `logout` ou `exit`.

3.4 Comandos Unix

Existem dois tipos de comandos no sistema Unix, os internos e os externos. Os internos são imediatamente executados pelo Shell, os externos necessitam da criação de processos que executam o código associado aos comandos.

Um comando Unix tem a forma geral seguinte: *c*omando opções argumentos
onde *o*pções tem a forma $\{\text{-}\{letra\}^+\}^*$.

Por exemplo, em `ls -s -i fich1 fich2`, `ls` é o nome do comando, `-s -i` as opções e `fich1 fich2` os parâmetros. Este comando pode também escrever-se `ls -si fich1 fich2`.

Qualquer comando tem de ser finalizado pelo caracter *newline* para ser executado pelo sistema. O valor de *newline* é `ctrl-j` (código ASCII 10). A forma mais simple de "validar"o comando é o envio do caracter `ctrl-m` (código ASCII 13) pela pressão da tecla `return` ou `enter`.

A execução de um comando lida com três ficheiros especiais, o de entrada, o da saída, e o da saída de erros. No caso standard, o teclado é a entrada e o ecrã os dois outros.

Um comando especial e muito importante é o comando `man`, que tem a sintaxe `man comando`. este comando disponibiliza uma ajuda em linha completa do sistema e das aplicações. Para saber como o `man` funciona, basta escrever `man man`.

3.5 Alguns comandos elementares

who O comando `who` escrito sem argumento devolve a lista dos utilizadores ligados ao sistema, com o argumento `am I`, o comando devolve a informação sobre o proprietário da sessão activa. Outros comandos mostram informação sobre a identificação de o (ou os) utilizador são `logname`, `id`, `rusers`, `finger`.

tty O comando `tty` (de *t*erminal type) devolve o nome completo da interface de conexão (interface virtual pelo qual o sistema comunica com o utilizador, visto que o mesmo utilizador pode estar conectado várias vezes no mesmo sistema)

echo O comando `echo` com um argumento de tipo "string"manda para a saída o referido argumento.

date Os comandos `date` e `cal`, devolvem informação sobre datas e a hora do sistema.

3.6 caracteres de controlo

A linha de comando do shell aceita alguns comandos de edição, como o **newline** que já encontramos. Resumidamente, são:

*c*aracter	*v*alor habitual	*e*feito por defeito
erase	`baskspace`	anulação do último caracter
kill	`ctrl-u`	anulação de uma linha
eof	`ctrl-d`	fim de introdução de dados
newline	`ctrl-j`	validação de um comando
intr	`ctrl-c`	interrupção
quit	`ctrl-\`	interrupção + imagem memória

O comando `stty` (set terminal type) com a opção `-a` permite visualizar a atribuição efectiva desses caracteres. Atribuir um novo valor a um caracter faz pelo mesmo comando, por exemplo: `stty erase £` indica ao seu terminal que o caracter de controlo `erase` é agora o `£`.

4 Sistema de ficheiros Unix

A noção intuitiva de árvores de ficheiros utilizada por exemplo em Dos para representar a hierarquia, ou mais simplesmente o sistema de ficheiros é também a representação lógica utilizada em Unix, mas com algumas diferenças essenciais.

Em Unix, tudo é ficheiro, e qualquer ficheiro tem uma só estrutura: uma sequência de caracteres não estruturada. Um ficheiro tem no entanto um tipo que define o conjunto das operações a que se pode sujeitar [2].

A um nível mais interno, num disco físico associa-se a cada ficheiro um bloco de informação, chamado **i-node** [3] que contem as informações seguintes:

- Número de identificação.
- Tamanho e Tipo de Ficheiro.
- Endereços dos blocos do disco onde está arquivado o ficheiro.
- Identificação do proprietário assim como as permissões de accesso.
- Número de links para o ficheiro.
- Outras informações, como a data de modificação...

Visto assim um ficheiro físico é anónimo, o nome não consta no i-node. Um nome é atribuido a um ficheiro somente ao nível dos directórios. Assim, um directório não é nada mais do que um ficheiro cujo conteúdo é uma lista de pares (nome,número) onde *n*ome é o nome que se atribui neste directório ao i-node de número *n*úmero (podemos considerar que um i-node é o representante de um dado ficheiro físico).

Nos termos da álgebra relacional, o atributo *n*úmero de identificação de um i-node é o campo chave da relação i-node. Assim num disco não existem dois i-nodes com o mesmo número, mas podem existir dois ficheiros (ou seja, dois pares *(nome,número)*) com o mesmo número i-node [4]. Cada par (nome,número) é chamado *l*ink para o ficheiro que o i-node de número *n*úmero representa. O campo número de links de um i-node é sempre actualizado pelo sistema e fornece então o número de link existentes. Quando este número atinge o valor 0, o sistema recupera o espaço físico ocupado pelo ficheiro no disco.

Esta propriedade de definição de varias referências para a mesma informação através dos links transforma a representação habitual do sistema de ficheiros em grafe acíclico dirigido cujos nodos são os indíces dos i-nodos contemplados e os vértices representam a hierarquia dos directórios e dos ficheiros. Cada vértice tem o nome que está associado ao i-node para o qual ele aponta. Por ilustração podemos tomar o exemplo da figura 1:

[2] Uma lista não exaustiva desses tipos pode ser: ficheiro disco, directório, ficheiro recurso sistema(device). Em particular qualquer periférico é visto como um ficheiro device. Assim uma impressora é geralmente representada por um ficheiro de nome `lpx` no directório `/dev`, onde x é o número da impressora (0 se for a única ou a primeira impressora)..

[3] i-node quer dizer *index node*

[4] Ou até com o mesmo nome, logo que não estejam no mesmo directório.

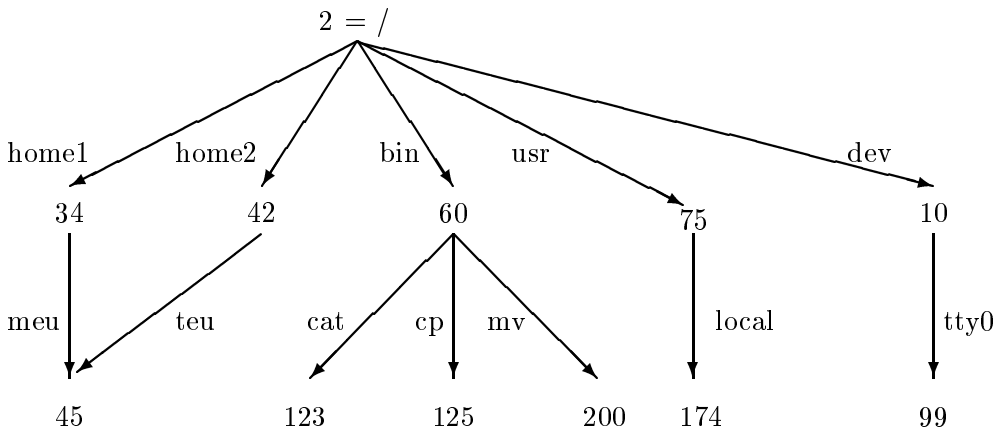

Figura 1: Exemplo de uma hierarquia de ficheiros

Notamos aí que `meu` e `teu` são dois nomes para o mesmo ficheiro (i-node 45).

Em qualquer hierarquia de ficheiro Unix destaca-se o directório principal a partir do qual todos dependem: o directório raíz, que é designado em Unix por o caracter "/", este tem por identificação o número 2. Destaca-se também a existência de dois ficheiros especiais em cada directório: `.` e `..` que representam sucessivamente o directório actual e o directório pai [5].

Ao nível da utilização do sistema, cada ficheiro tem um nome. Alcançar um dado ficheiro pode ser realizado de duas formas:

Por referência absoluta Qualquer ficheiro ou directório pode ser designado pelo caminho que permite passar da raíz até a sua localização. No exemplo da figura 1, a referência absoluta do ficheiro `mv` é `/bin/mv`.

Por referência relativa A qualquer instante um utilizador conectado está a trabalhar num directório. Este directório é designado por *d*irectório de trabalho. Qualquer ficheiro pode ser designado relativamente a esse directório. Se estivermos no directório `bin`, a referência relativa do ficheiro `mv` é `mv` e no caso do directório de trabalho ser `home2` temos `../bin/mv`. Outro mecanismo de referência relativa é usar os mecanismos de substituição (ver secção 4.4).

4.1 Montagem de sistemas de ficheiros

Consideramos até aqui que o sistema contemplava um só disco físico. No sistema Unix, cada disco físico, e mesmo as suas eventuais partições lógicas, tem o seu próprio sistema de ficheiros. Um tal disco pode ser "montado"num outro. O que significa que o disco montado é visto como um directório do disco onde foi montado, e isto de forma totalmente transparente para o utilizador. Obtemos assim um só sistema de ficheiros para todo o sistema, qualquer que seja a sua estrutura física. Um inconveniente desta facilitade Unix consiste numa restrição ao mecanismo de link. Isto porque pode existir num sistema de ficheiros montado um i-node com o mesmo indíce que um i-node do sistema de ficheiros "pai". A linkagem de um ficheiro entre dois sistemas de ficheiros montados um no outro realiza-se então pela criação de um ficheiro particular chamado *l*ink simbólico, no local escolhido para o link. O conteúdo desse ficheiro será então a referência do i-node desejado.

[5] Como em DOS.

4.2 Propriedades de um ficheiro

Em cada i-node estão especificadas as propriedades do ficheiro em termos de política de segurança. Além da identificação do proprietário, assim como do grupo a que ele pertence, as propriedades podem ser agrupadas em três grupos:

- Propriedades do proprietário do ficheiro (`u`)
- Propriedades do grupo proprietário (grupo ao qual o proprietário pertence) (`g`)
- Propriedades dos outros utilizadores (`o`).

Para cada um desses grupos estão definidos os direitos de:

- leitura (`r` ou `4`)
- escrita (`w` ou `2`)
- execução (`x` ou `1`)

O direito de execução para um directório consiste no direito de fazer desse directório um directório de trabalho, a leitura consiste na capacidade de listar o conteúdo do directório, a escrita na capacidade de remover ou acrescentar ficheiros.

A informação sobre o tipo de ficheiro é também contida na lista das propriedades (`-` para ficheiro de dados, `d` para directório, `c` e `b` para os ficheiros especiais).

Em resumo, estão presentes nos i-nodes as propriedades seguintes: nome do proprietário, nome do grupo proprietário, e 10 bits de informação. O primeiro bit para o tipo do ficheiro, os três seguintes para os direitos de leitura escrita e execução do proprietário, os três outros para os do grupo proprietário, e os três últimos para o resto dos utilizadores.

4.3 Comandos do sistema de ficheiros

mkdir *n*ome : Permite a criação do directório *n*ome no directório de trabalho.

rmdir *n*ome : remove o directório *n*ome se ele está vazio e presente no directório de trabalho.

pwd : devolve o caminho absoluto do directório de trabalho.

cd *n*ome : desloca o directório de trabalho para o directório *n*ome se este está presente no actual directório de trabalho. `cd` sem argumentos desloca o directório de trabalho para a "home directory"do utilizador.

ls *n*ome : lista o conteúdo do directório *nome* se esse existir. Opções do comando permitem mostrar mais informação relativamentes ao ficheiros listados, assim como listar ficheiros escondidos.

cp *n*ome1 *n*ome2 : realiza uma copia física do ficheiro *n*ome1 para *n*ome2. O que acontece realmente é a criação de um i-node e de um par adequado no directório destino.

mv *n*ome1 *n*ome2 : Modifica o conteúdo e/ou a localização do par *(*nome,número) presente no directório apontado por *n*ome1. Se não houver mudança de directório entre *n*ome1 e *n*ome2, `mv` contenta-se em mudar o nome no par, senão `mv` desloca o par para o referido directório, mudando se necessário o nome do ficheiro.

ln *n*ome1 *n*ome2: acrescenta um link com o nome e no directório referenciados em *n*ome2 ao i-node referenciado por *n*ome1.

cat *n*ome : visualiza o conteúdo completo do ficheiro referenciado por *n*ome.

more *n*ome : visualiza um ficheiro *n*ome, página por página. As principais opções são as seguintes: `h` para a ajuda, `f` para avançar, `b` para recuar, `/motivo` para a procura de `motivo`, `n` para repetir a procura, `q` para sair.

rm *n*ome : apaga o par (nome,número) do directório contemplado, e decrescenta de uma unidade o número de links sobre o i-node de número *n*úmero.

chmod : com as devidas opções e parâmetros permite mudar os direitos de acesso de um ficheiro. O utilizador tem de ter direitos suficientes para realizar esta operação.

chown : com as devidas opções e parâmetros permite mudar o proprietário de um ficheiro. O utilizador tem de ter direitos suficientes para realizar esta operação.

du *n*ome : mostra o tamanho da hierarquia do sistemas de ficheiros abaixo do directório *n*ome.

df : mostra os valores da utilização do disco.

4.4 Mecanismos de substituições

A semelhança de muitos sistemas operativos, o sistema Unix oferece mecanismos de substituições:

***** : significa *q*ualquer sequência de caracteres. `ls ab*.*` tem por efeito a listagem de todos os ficheiros cujo nome começa por `ab` e que posuiem um extensão.

? : significa *q*ualquer caracter. `ls ab?.??` tem por efeito a listagem de todos os ficheiros cujo nome começa por `ab` e de comprimentos 3, e que tem uma extensão de dois caracteres.

[$\cdots$] : `ls [a-z][a,b].[!c]` tem por efeito a listagem de todos os ficheiros cujo o nome tem um comprimento de dois caracteres cujo o primeiro caracter e uma letra minúscula, cujo o segundo ou é `a` ou é `b` e que tem por extensão um único caracter que não é o caracter `c`.

~ : este símbolo é utilizado como atalho para o directório pessoal de utilizadores. Sozinho representa o directório pessoal do utilizador activo, com a indicação de um login name ele indica o directório do utilizador referenciado. Por exemplo `~desousa/bin/sc` é uma referência relativa do ficheiro `sc` que está no directório `bin` do directório pessoal do utilizador `desousa`.

5 Processos, redireccionamentos e pipes

Em Unix, vários programas podem ser concorrentemente executados. Assim sendo, o sistema terá de alocar o processador de forma equitável a cada um dos programas. A parte do sistema que se encarregua desta partilha de tempo é o *s*cheduler. Esta noção de concorrência implica certas distincções importantes: temos de distinguir a noção de programa da noção de processo, tal como discriminar vários tipo de programas. Um programa é (e não passa de) uma sequência de instruções arquivadas em geral num ficheiro. Distingue-se:

- Os ficheiros **fonte**: são os ficheiros de texto escritos numa linguagem de programação.

- Os ficheiros **binários** ou **executáveis**: são os resultados da compilação dos ficheiros fonte. Os binarios podem ser executados pelo processador.

- Os ficheiros de **comandos**: os já referidos shell scripts que contêm comandos que o shell vai interpretar e mandar para o processador.

Podemos então considerar um programa como um "plano de acções"que o sistema terá de seguir para chegar ao resultado requerido. A noção de programa é por isso estática. Ora a execução de um programa exige a alocação dinámica de recursos do material [6]. É a essa representação dinámica que chamamos **processo**: um processo representa um programa assim como o estado em que se encontra a sua execução. Umas das razões da existência de tal distincção reside na vertente multi-tarefa do sistema Unix. A execução de um programa não é contínua, e pode então ser interrompida a qualquer momento e retirada da memória para deixar que outro programa se execute. O sistema tem de ser então capaz de retomar a execução do processo no estado em que o deixou.

Assim e de forma simplificada, um processo tem por atributos um número identificador(chamado `Pid`, para *P*rocess Identification), o programa que ele corre, o estado da sua execução, o estado em que o processo se encontra [7], tempo de execução, identificação do processo "pai", nome do proprietário, etc...

5.1 Gestão dos processos

Um processo representa a execução de um programa, mas este programa pode mandar executar de forma concorrente um programa auxiliar. A execução deste programa auxiliar é então atribuída a outro processo. Este novo processo depende do primeiro. Diz-se que é filho deste. Gera-se assim um verdadeira genealogia de processos [8].

Para um bom encaminhamento dos processos todos, o sistema Unix utiliza algo semelhante ao sistema de ficheiro. Assim o i-nodo de um processo é chamado **bloco de controlo**. O conjunto dos blocos de controlo são arquivados numa tabela de processos.

5.2 Comandos sobre processos

ps : `ps` para *p*rocess status; mostra a lista de processos pertencendo a um conjunto particular, assim como algumas das suas características, consoante as opções. É o equivalente do comando `ls` mas para os processos. Por defeito, isto é sem opções, o comando mostra quatro itens de informação: o Pid, o terminal de execução (tty), o tempo acumulado de execução (tempo que o processador passou a executar o processo e a sua descendência), e a identificação do programa associado ao processo. Obter uma lista de mais processos, assim como de mais informação sobre os processos, requer o uso das opções.

kill -sinal identprocesso: Envia o sinal *s*inal ao processo de número *i*ndentprocesso. Um uso clássico do comando é `kill -9 x` onde `x` é o Pid do processo contemplado. Este comando mata o processo (muito útil no caso de o processo estar "encravado"). `kill -l` lista todos os sinais possíveis (num formato alfanúmerico, e não númerico).

[6] Memória para as variáveis, para o empilhamento de funções, ficheiros temporários, etc...

[7] Não entrando em pormenores, o estado de um processo indica se o processo está activo (a ser executado), adormecido, morto etc...

[8] Podemos assim representar o conjunto dos processos por uma árvore "geneológica"de processos.

at *momento* : lê na entrada standard os comandos que serão executados no momento *momento*. Uma variante é `at -f fich momento`, que indica ao sistema que os comandos presentes no ficheiro `fich` terão de ser executados ao `momento` indicado. Por exemplo `at 14:45 < ls`.

5.3 Ficheiros standard e redireccionamentos

A cada momento de uma sessão Unix, um utilizador dispõe de três ficheiros especiais que representa os recursos de comunicação pelos quais os comandos vão comunicar:

entrada standard : por defeito o teclado. É nesse ficheiro que serão lidas as informação de entrada (os inputs).

saída standard : por defeito o ecrã. Serão aí colocados os dados de saída (os outputs).

saída de erros standard : por defeito o ecrã. Qualquer mensagem de erro será enviado para esse ficheiro.

É no entanto possível, e mesmo de uso comum, modificar os valores por defeito. Chamamos essas modificações **redireccionamentos**. As duas formas usuais de redireccionamento são o uso de opções nos comandos usados que permitam o redireccionamento desejado, ou o redirecionamento manual pelo uso de operadores especiais que se acrescentam ao envio do comando ao sistema:

comando **>** ***fich*** : redirecciona a saída standard para o ficheiro *fich*.

comando **>>** : ***fich*** : redirecciona a saída standard para o ficheiro *fich*. Concatena a saída standard ao ficheiro *fich* se esse já existir.

comando **<** ***fich*** : redirecciona a entrada standard para o ficheiro *fich* se *fich* for um ficheiro.

comando **2> ou >&** ***fich*** : redirecciona a saída de erros standard para o ficheiro *fich*.

comando **2>> ou >>&** ***fich*** : redirecciona a saída de erro standard para o ficheiro *fich*. Concatena a saída de erros standard ao ficheiro *fich* se esse já existir.

5.4 Encadeamento de processos

Outra facilidade do sistema Unix é a combinação de processos. Podemos combinar de três formas diferentes os procesos Unix:

; : `comando1 ; comando2` executa `comando2` logo a seguir ao `comando1`. Trata-se dum operador de **sequência**.

| : É por esse operador que se define a execução concorrente de processos por **pipe**.
`comando1 | comando2` tem por acção a execução em paralelo dum processo P_1 associado ao comando1 e dum processo P_2 associado ao comando2. A entrada standard de P_2 está redireccionada para a saída standard de P_1. A zona de comunicação criada entre os dois processos é então gerida por mecanismos clássicos de *p*ilhas, sendo a execução dos processos concorrente e não sequencial. A execução por pipe de processos é muito útil quando queremos executar um programa que utiliza os dados que outro cria. Por exemplo: `ls -al | more`

& : Qualquer processo lançado pelo utilizador é filho do processo que representa o shell de onde se lançou o comando. Enquanto este processo não terminar, o shell, sendo bom pai, vai esperar pelo filho. Existe uma maneira de forçar o shell a não esperar pelo fim do processo filho, de emancipar o processo do shell que o lançou. A autonomia dum processo é garantida pelo o uso, em fim de comando, do operador &. Diz-se que o processo foi executado em **background**. Por exemplo `ps -aux &`. A vantagem da execução em background é a possibilidade de continuar a trabalhar com o shell, apesar do processo criado ser demorado.

Como em aritmética, existem prioridades de "operadores". Forçar uma interpretação especial de um comando pode requerer parênteses. Por exemplo

`date ; who > fich` $\neq$ `(date ; who)>fich`

6 Filtros

Um processo está muitas vezes associado a execução de um programa que cria resultados em formato ficheiro. É muito útil ter então mecanismos de vizualização que permitam filtrar o resultado dos processos, isto porque os resultados vem em quantidade enorme, ou de forma desordenada, inadequada, ou ainda com alguma informação indesejada. Esses mecanismos formam o que chamamos **filtros**. Os filtros são usados com a ajuda dos mecanismos de pipe, de redirecionamento, ou directamente pelas opções dos filtros.

cat : permite mostrar o conteúdo completo da entrada standard, ou do ficheiro parâmetro.

wc : devolve o número de linhas, palavras e caracteres da entrada standard. Opções do comando permitem devolver só partes dessa informação.

head -n : mostra as n primeiras linhas da entrada.

tail n : mostra as últimas linhas da entrada a partir da linha n (opção +), ou mostra as n últimas linhas (opção -).

tr *"string1" "string2"* : *s*tring1 e *string2* têm o mesmo tamanho. Transforma a entrada substituindo cada caracter da entrada presente na *s*tring1 pelo caracter corespondente da *s*tring2. Por exemplo `(tr "ab"`*`"12"`*` < fich) > fich2` transforma todos os `a` de `fich` em `1` e os `b` em `2`, sendo o resultado colocado no ficheiro `fich2`.

sort : ordena de forma lexicográfica a entrada.

grep *"string"*: este comando é muito completo, o uso habitual dele consiste num parámetro *"string"*, o efeito é de selecionar as linhas da entrada que contem a expressão *"string"*.

more : vizualiza a entrada página por página.

cut : permite a seleção de colunas (em oposição às linhas selecionadas pelo `grep`), ou de campos de um dado ficheiro. Por exemplo `cut -d: -f1,5 /etc/passwd` coloca na saída a infomação *l*ogin_name : nome para cada entrada do ficheiro `/etc/passwd`, ou seja para cada utilizador do sistema.

Existe o programa Unix `awk` que consegue de uma forma muito eficiente e completa, mas de forma "complicada", servir de filtro. `awk` é uma verdadeira linguagem de programação sobre ficheiro que permite filtrar qualquer tipo de informação através mecanismos de ciclos, variáveis, expressões regulares, etc... Aconselhamos assim ao leitor uma visita ao `man awk`.

7 Comandos úteis

Damos nesta secção uma lista de comandos de uso comum. Limitamo-nos aqui a explicar o funcionamento geral desses comandos, deixando ao leitor o cuidado de consultar a página de manual correspondente (`man comando`).

compress, gzip : programas de compressão. Ex: `gzip ficheiro`

uncompress, gunzip : programas de descompressão. Ex: `gunzip ficheiro.gz`

tar : Arquiva num ficheiro vários ficheiros. Este programa também realiza a operação inversa. Ex: `tar -cvf fich.tar *.*`, arquiva todos os ficheiros do directório de trabalho no ficheiro `fich.tar`. A operação inversa é realizada por o comando `tar -xvf fic.tar`. Costuma-se usar este comando em conjunto com o comando `gzip` (ou `gunzip`) para fazer ou desfazer backups de ficheiros.

diff : Vizualiza as diferenças entre os dois ficheiros dados em parâmetro.

lp ou lpr : `lpr fich` manda o ficheiro `fich` para a impressora. Ex: `lp fich`, cria um pedido de impressão para a impressora por defeito, com a identificação `x`.

lpstat ou lpq : vizualiza informação sobre o estado das impressões e da impressora.

cancel, lprm : permite cancelar pedidos de impressão ainda em fase de espera (visíveis pelo comando `lpstat`). Ex: `cancel x` cancela a impressão que tem por identificação `x`.

touch : modifica a data de última modificação do ficheiro dado em parâmetro. É criado o ficheiro em parâmetro se esse não existir,

dirname : devolve a parte nome de directório de uma referência de um ficheiro dada em parâmetro.

basename : devolve a parte nome de ficheiro de uma referência de um ficheiro dada em parâmetro.

find : Este comando completo permite uma procura em disco. Ex: `find / -name fich -print` procura a partir da raiz (`/`) o ficheiro de nome `fich` e vizualiza as referências encontradas.

chsh : permite modificar o shell de lançamento ao iniciar uma sessão Unix. De facto permite a modificação dum campo das declarações relacionados com o utilizador no ficheiro `/etc/passwd`. Ex: `chsh desousa /bin/bash` tem por efeito de inicializar o campo shell do utilizador `desousa` no `/etc/passwd` do sistema para `/bin/bash`. A lista dos shells acessíveis encontra-se geralmente num ficheiro chamado `/etc/shells`.

id : devolve a identificação sistema do utilizador activo (o *u*ser id, o *g*roup id de que é membro).

login : permite iniciar uma nova sessão Unix a partir de uma sessão já aberta.

passwd : permite modificar a senha do utilizador activo. Para isso o utilizador deverá introduzir a antiga senha e duas vezes a nova senha.

logname : devolve o nome do utilizador (que está ligado ao user id). Este comando consulta simplesmente o ficheiro `/etc/passwd`.

tty : vizualiza a informação sobre o terminal usado na sessão activa.

finger *n*ome: vizualiza informação geral sobre o utilizador, que tem por login name *n*ome. Por exemplo informação sobre o nome real do utilizador, que shell costuma usar, a data da última conexão, se tem mail ou não, etc... Esta informação tem de forma geral sempre um conteúdo pessoal, e representa a Informação pública que um dado utilizador disponibiliza ao resto dos utilizadores [9]. Informação adicional pode ser fornecida pelo utilizador pelo intermédio dos ficheiros `.project` e `.plan` presentes na sua home directory. Este comando pode também ser utilizados sem parâmetrro.

who : devolve a lista dos utilizadores ligados ao sistema, assim como os terminais utilizados e a hora de ligação [10].

date : devolve a data e hora actual do sistema.

cal : comando de consulta do calendário do sistema. Ex: `cal 2333` vizualiza o calendário do ano 2333.

echo : escreve na saída standard os seus parâmetros. Ex: `echo Bom Dia $USER` tem por efeito, se supormos que `USER` é uma variável que contém o nome do utilizador, vizualizar a frase "Bom Dia utilizador".

clear : limpa o ecrã.

mail : permite a escrita de correios electrónicos, assim como a sua leitura. O ficheiro `.mailrc` contém a configuração deste utilitário. Ex: `mail desousa@noe.ubi.pt` permite a escrita de um mail dirigido ao `desousa@noe.ubi.pt`. A escrita de um mail deve ser concluida por um ponto final em início de linha. Este sinal será interpretado pelo sinal como ordem de envio da mensagem escrito para o destinatário. O comando `mail` sem argumentos coloca o utilitário em modo leitura, se houver mail para ler. Neste caso a opção `h` permitira vizualizar um menú de ajuda. Como no caso do comando `finger`, é possível deixar no fim de cada mansagem uma *a*ssinatura pessoal pelo intermédio da criação, na home direcctory, do ficheiro `.signature` que contém a referida "mensagem assinatura" [11].

whereis : explora a variável `PATH` a procura do programa dado em parâmetro e devolve as diferentes referências encontradas assim como as fontes do programa e as páginas de manual.

which : explora a variável `PATH` a procura do programa dado em parâmetro e devolve as diferentes referências encontradas.

8 Introdução aos shells

8.1 O que é um shell?

Um shell é uma caixa de ferramentas do sistema de alto nível que nos permite disfrutar de toda a riqueza do sistema Unix. Sobrepõe-se ao sistema para que certas facilidades sejam disponibilizadas

[9] A informação vizualizada é na sua maioria informação encontrada no ficheiro `/etc/passwd`.

[10] Ao contrário do comando `finger`, a informação aí vizualizada tem um conteúdo mais técnico e está mais ligada ao ambiente sistema em que está envolvido o utilizador. Basiea-se essencialmente na consulta de um ficheiro sistema `/etc/utmp` mantido e actualizado pelo próprio sistema.

[11] Esta possibilidade merece uma olhada aprofundida ao `man mail` para que possa ser adequadamente realizada.

e que qualquer pedido do utilizador seja da melhor forma apresentada ao sistema. Esses pedidos podem ser apresentados de duas formas, pela introdução directa ao teclado, ou pelo intermediário de um ficheiro texto executável chamado **shell script**.

Para esse efeito os shells apresentam-se em dois níveis estruturais:

Interpretação dos comandos : Encontramos aí diversos mecanismos de substituições, por exemplo os wildcards, aliases, compleção de nomes de ficheiros, mecanismos de histórico de comandos, mecanismos de variáveis, etc...

De facto podemos atribuir três funções ao *i*nterpretador de comandos do shell:

1. O modo interactivo: Consiste numa interface entre o utilizador e o sistema. Permite submeter ao sistema (pelo intermédio do nível estrutural seguinte) tarefas do utilizador. Intervêm aí os mecanismos básicos de substituição.
2. A gestão do ambiente de trabalho: O interpretador de comandos deve poder realizar uma gestão do ambiente de trabalho do utilizador, que consiste essencialmente em operações sobre variáveis do sistema (ou do shell), ou ainda variáveis definidas pelo próprio utilizador.
3. A programação shell: Para tarefas complexas ou compostas, os mecanismos básicos de substituição podem não ser suficientes. Para completar as funcionalidades do shell é disponibilizada uma autêntica linguagem de programação sistema.
   Assim, em shell `sh`, o comando: `for i in 1 2 3 ; do echo ‘date‘; done` é interpretado numa primeira fase [12] em
   ```
   echo ‘date‘
   echo ‘date‘
   echo ‘date‘
   ```
   e numa segunda fase [13] em
   ```
   echo Thu Mar 12 16:17:47 WET 1998
   echo Thu Mar 12 16:17:47 WET 1998
   echo Thu Mar 12 16:17:47 WET 1998
   ```
   Esses três comandos são finalmente submetidos ao segundo nível estrutural do shell.

Execução de comandos interpretados : A execução é realizada de duas formas, consoante a natureza do comando:

- Os comandos internos, como `pwd`, `echo` ou `cd`, são directamente executados pelo shell porque correspondam a funções internas do shell.
- Os comandos externos, como `ls`, `cat` ou `mv`, são enviadas como processos ao sistema pelo shell. O shell tem de ser capaz de invocar os programas relacionados de forma eficiente. Para isso tem de ser capaz de usar e fornecer mecanismos de gestão de processos: o **job control**.

8.2 Execução de um shell

Existem vários shells disponíveis num sistema Unix [14]. Dividem-se em duas famílias, a família dos bourne shells (sh, bash, ksh...) e dos C-shells (csh, tcsh...). As diferenças residem essencialmente

[12] Esta fase corresponde a interpretação do programa shell.

[13] Esta fase corresponde na substituição da string date pelo resultado do comando `date`, fase que podemos ligar aos mecanismos básicos de subtituição.

[14] Consulte o ficheiro `/etc/shells`.

ao nível da interactividade e da linguagem disponibilizada.

8.2.1 Login shell

O processo de conexão disponibiliza para cada utilizador um shell "inicial", chamado **login shell**, que se encareguera de toda a gestão da sessão do utilizador. A escolha deste shell está definida no último campo da entrada do utilizador no ficheiro `/etc/passwd`.

Correr outro shell, ou mudar de login shell pode ser efectuado de pelo menos três formas:

1. Execução directa pela referência do binário associado ao shell. Por exemplo `/bin/sh` executará o bourne shell. Este shell não substitui o login shell, sobrepõe-se a ele (para ser mais correcto, o shell criado torna-se filho do login shell).

2. A substituição "real"do login shell por outro realiza-se pelo comando `exec` *n*ovo_shell em que *novo_shell* é a referência do shell pretendido.

3. Pode-se também modificar o campo do ficheiro `/etc/passwd` para que a próxima conexão corra um outro login shell. O comando a utilizar é o já descrito `chsh`.

8.3 Metacaracteres

Para que as funcionalidades do shell sejam utilizadas pelo utilizador, foi necessário recorrer a especialização de certos caracteres, ou seja de atribuir accões especiais a esses caracteres. Estes caracteres são chamados **metacaracteres**:

- Introdução de comentários: `#`
- Operador de despecialização de metacaracteres: `\` (antislash)
- Operadores de delimitação de strings: `' " ‘`
- Operadores de redireccionamentos: `< > >> >& >>&`
- Operadores de composição de comandos: `; | &`
- Operador de substituição variáveis: `$`
- Wildcards: `* [ ] ! - ?`
- Operadores de histórico de comandos: `! - :`
- Operadores de compleção de nome de ficheiros: `TAB`

8.4 Mecanismos de substituição

Já foram espostos a maior partes desses mecanismos. Contentamo-nos aqui de completar a lista desses mecanismos.

8.4.1 Variáveis

Num shell, uma variável é identificada por uma sequência de caracteres alfanumericos, começada necessariamente por uma letra (_ é considerada letra).

Podemos classificar as variáveis shell em três grupos:

1. Variáveis que desenvolvem um certo papel no comportamento do shell, mas que podem ser eventualmente ser alterados pelo utilizador.
2. Variáveis internas ao sistema, que são completamente geridas pelo shell mas que podem ser consultadas. Variáveis ditas de **posição** são as que correspondem aos argumentos dos comandos interpretados.
3. Variáveis definidas pelo utilizador.

Outra classificação das variáveis é:

- Variáveis locais ao shell. São variáveis que não são transmitidas à descendência (aos processos filhos).
- Variáveis de ambiente ou **exportadas**. Ao contrário das variáveis locais, essas variáveis são transmitidas a descendência, como por exemplo as aplicações lançadas ou os shells "filhos".

O tipo de base das variáveis é o tipo "string", mas é possível extender o tipo das variáveis aos vectores e até aos números.

O mecanismo de substituição baseia-se no metacaracter `$`. Assim $*v*ariável ou ${variável} devolve o valor associado a *v*ariável.

8.5 Delimitar sequências de caracteres

´ : todos os caracteres da sequência delimitada por ´ são considerados como protegidos, isto é, perdem o caracter especial, se tiverem um. Assim ´`date` ´ é considerada como a sequência `d a t e` .

¨ : no interior deste delimitador, os caracteres `\ ’ ‘ ! $` ainda guardam o significado especial.

` : uma sequência delimitada por esse caracter é interpretado como um comando. Assim ‘`date`‘ é substituido pela sequência `Thu Mar 12 16:17:47 WET 1998`, outro exemplo é ¨`echo ` `date` `¨ que é substituido pela sequência `echo Thu Mar 12 16:17:47 WET 1998`

8.6 Job Control

O que chamamos processo ao nível do sistema Unix corresponde a um job (ou tarefa) numa visão local ao shell. Cada shell tem de poder gerir o seu conjunto de jobs e disponibiliza para isso algumas ferramentas que agirão ao nível do shell e não ao nível do sistema. Podemos citar por exemplo o comando `ps` que é um comando de gestão de processo ao nível do sistema e não ao nível do shell. Este comando é bastante mais pesado do que os comandos que agem somente ao nível do shell. Esta gestão complementa a zona bloco de contrôlo dos processos do sistema. Assim o *P*Id ao nível sistema é segundado ao nível do shell por uma identificação delimitada pelo caracter `%`. Em particular atribui-se a cada job criado um número. Resume-se este mecanismo de indentificação por:

%*n*úmero	Job de número local *n*úmero
%*s*tring	Job cujo nome começa por *s*tring
%?*s*tring	Job cujo nome contêm *s*tring
%%	Job "corrente", identificado pelo sinal + na lista de jobs
%+	Equivalente ao %%
%-	Job "corrente"precedente (indentificado por -)

A visualização da lista dos jobs é feito pelo intermédio do comando shell `jobs`, com eventualmente a opção `-l`.

Porque qualquer shell tem uma gestão eficiente e sem ambiguidades dos jobs, o comando `kill` aceita a identificação por `%`

Imaginemos que `ciclo` seja um programna que não pára. Podemos então ter a sessão Unix sguinte:

```
$ ciclo&
[1] 898
$ ciclo&
[2] 899
$ ciclo|ciclo&
[3] 900
$ ps|grep ciclo
    899 tty2    0:00    ciclo
    900 tty2    0:00    ciclo
    901 tty2    0:00    ciclo
    898 tty2    0:00    ciclo
$ jobs -l
[3]     +       900     Running     ciclo|ciclo&
[2]     -       899     Running     ciclo&
[1]             898     Running     ciclo&
$ kill %3
[3]     +       Terminated  ciclo|ciclo&
$ jobs -l
[2]     -       899     Running     ciclo&
[1]             898     Running     ciclo&
$
```

Cada job a qualquer momento encontra-se em três estados possíveis (dois dos quais já foram referidos).

1. em primeiro plano (*f*oreground)
2. em *b*ackground
3. suspenso. Este estado é o estado dos processos parados que estão eventualmente a espera de serem retomados.

A passagem de um estado para o outro é feito da seguinte forma:

foreground-suspenso : `ctrl-z`

suspenso-foreground : `fg` *i*`dent_tarefa`

background-suspenso : qualquer tentativa de leitura na entrada standart pelo processo em background, ou explicitamente por `stop` *i*`dent_tarefa`

suspenso-background : `bg` *i*`dent_tarefa`

background-foreground : `fg` *i*`dent_tarefa`

8.7 C-shell

8.7.1 Generalidades

Ficheiros de configuração :
Quando é lançado um C-shell são consultados dois ficheiros de inicialização:

1. Se o C-shell for um login shell, é consultado um ficheiro de configuração sistema (comum a todos os utilizadores) `/etc/profile`.
2. Se existir, é consultado em todos os casos (login shell ou não) o ficheiro `.cshrc` situado no directório privado do utilizador. As configuração aí colocadas são configurações próprias do utilizador.

Fecho dos processos csh :
Os seguintes comandos permitam acabar com os processos `csh`: `ctrl-d`, `logout`, `exit`.

Execução de ficheiros de comandos :
Um ficheiro texto pode ser submetido ao shell das formas seguintes:

Comando	**Condição sobre *r*ef**	**Efeito**
`csh` *r*ef	ficheiro legível	interpretação por um subprocesso `csh`
*r*ef	ficheiro legível e executável cuja primeira linha comaça por `#!` *r*ef_sh	interpretação por um subprocesso denome *r*ef_sh
*r*ef	primeira linha contendo `#`	subprocesso `csh`
*r*ef	primeira linha não contendo `#`	subprocesso `sh`
`source` *r*ef	ficheiro legível	interpretação pelo shell corrente
`exec` *r*ef	ficheiro legível e executável	Cobertura do shell corrente por um `csh` interpretando *r*ef

8.7.2 Variáveis

O C-shell autoriza além de strings, a manipulação de vectores e a evaluação de expressões aritméticas.

Mecanismos de Substituição :

$*variável* : é substituido pelo valor de *v*ariável.

$*variável*[*índice*]: índice é ou um número ou um intervalo da forma $i - j$ ou * (para todos os índice). é substituido pela lista dos elementos referidos pelo índice.

$#*variável* é substituida pelo número de palavras da variável.

$0 : nome do ficheiro executado.

$*inteiro* : valor do parâmetro de número *i*nteiro.

$* : todos os parametros.

$?*variável* : devolve 0 ou 1 consoante a existência de *v*ariável.

$$: número do processo.

Comandos de manipulação :

set *variável* : define a variável *v*ariável (valor=vazio).

set *variável*=*string* : define localmente *v*ariável como tendo por valor *s*tring.

setenv *variável* [*string*]: define *v*ariável como variável de ambiente com o valor (opcional) *s*tring.

unset *variável* : remove a definição da variável local *v*ariável.

unsetenv *variável* : remove a definição da variável de ambiente *v*ariável.

printenv : lista as variáveis de ambiente assim como os respectivos valores.

@ *variável*=*expressão*: define *variável* como tendo por valor a evaluação da expressão aritmética *e*xpressão.

set ou @: lista as variáveis locais assim como os respectivos valores.

Variáveis predefinidas :
Eis aqui a lista de algumas variáveis locais úteis:

argv : lista de argumentos do shell.

cwd : referência absoluta do directório de trabalho.

home : referência absoluta do directório privado.

path : lista de directórios de pesquisa para a procura de comandos.

prompt : string representando o prompt.

shell : referência absoluta do shell.

status : código de erro do último comando utilizado.

term : tipo de terminal usado

history : se for definida, esta variável activa os mecanismos de histórico. O seu eventual numérico valor define o número de comandos para memorisar.

filec : se for definida, esta variável permite facilidades de compleção de nome de ficheiros.

Eis aqui a lista de algumas variáveis de ambiente úteis.

HOME : valor deduzido de `home`.

LOGNAME : nome do utilizador.

PATH : valor deduzido de `path`.

SHELL : valor deduzido de `shell`.

TERM : valor deduzido de `term`.

Expressões :
Damos aqui uma pequena lista de operadores que podem ser incluídos numa expressão, ordenados por prioridade:

- () parêntesis.
- ! negação lógica.
- % modulo, / divisão, * multiplicação.
- - substracção, + soma.
- > maior, < menor, >= maior ou igual, <= menor ou igual.
- == igualdade, != diferença.
- && conjunção lógica, || disjunção lógica.

Relembramos que *falso* é o valor 0, e que 1 é o valor *verdade*

8.7.3 Mecanismos de Base

Outro tipo de expressão a referir é o tipo dos testes que em C-shell tem a forma seguinte: `-especificação referência`, onde `especificação` pode ser:

d	tipo de directório
e	existência do ficheiro
f	tipo ordinário
o	proprietário do ficheiro
r	direito de leitura
w	direito de escrita
x	direito de execução
z	tamanho nulo

Mecanismo básicos :
Falta nos referir algumas facilidades "exóticas"que o C-shell disponibiliza:

redirecção : a saída de erro é `>&`

alias : é possível fazer uns álias (dar um nome a) a comandos complicados, ou frequentemente utilizados. Temos assim a possibilidade de referir por um nome simples um comando que não o é necessariamente, isto é, definimos um sinónimo. Basta, para esse efeito, escrever (ou colocar no ficheiro `.cshrc`) o comando `alias` *`nome comando`*. o comando que anula um alias é `unalias` *`nome`*

histórico : se a variável `history` estiver definida, o C-shell disponibiliza o mecanismo de histórico que consiste em poder editar e chamar antigos comandos. Neste caso o comando `history` devolve a lista dos comandos memorizados previamente numerados. Executar um comando da lista faz-se pelo comando `!`*`número`*. Para ilustrar umas facilidades suplementares, damos o exemplo seguinte: `!38:0-5:s/a/qwerty`. O efeito é chamar o comando 38 em que só são tomados em conta os parâmetros 0 (o nome do comando) à 5 e nesses parâmetros, qualquer string "a"é substituido por "querty".

compleção de nome de ficheiro : no caso da variável `filec` ser definida, é possível pedir ao shell que completa, pela pressão da tecla `TAB`, nomes de ficheiros.

8.7.4 Procedimentos C-shell

Parâmetros :
Relembramos que `$argv` representa a lista dos argumentos argumentos dum comando(ou `$*`), `$argv[n]` o n-ésimo argumento (outra forma equivalente é `$n`), `$#argv` o número de argumentos.

Linguagem :

Labels : é possível definir etiquetas num shell script da forma seguinte: `nome:`. O comando `goto nome` permite a deslocação para a etiqueta `nome`.

condicionais : a forma mais geral do if é:

```
if (expressão) then
comandos
else if ···
...
endif
```

mas que podemos simplificar por

```
if (expressão) comandos
endif
```

É também possível definir condicionais múltiplas através o `switch`

```
switch (valor)
case v1:  comandos
breaksw
...
```

```
case vn:  comandos
breaksw
default:  comandos
endsw
```

ciclos :

`repeat` *inteiro* `comando`
Permite repetir *inteiro* vezes o *comando*.
`foreach` *variável* `(`*lista de valores*`)`
comandos
`end`
Permite repetir para os valores da lista que *variável* percorre, os *comandos*.
`while (`*expressão*`)`
comandos
`end`
Permite executar *comandos* até *expressão* se tornar falsa.
`break` permite sair de o ciclo em que se encontra.

Referimos aqui um comando útil para a escrita de shells script é a função `exit (n)` que permite concluir um programa e devolver n como código de retorno (ou código de erro).

8.7.5 Exemplos

A título de exercício analise e explique os dois scripts seguintes:

```
%cat indice
#indice de variavel
set n=2
while ($n <= $#argv)
    if ($argv[1] == $argv[$n])
         then
       @ n--; echo indice $n; exit $n
     else
       @ n++
    endif
echo elemento nao encontrado
exit (0)
%
```

```
%cat datapt
#em portugues!
set ldi = (Mon Tue Wed Thu Fri Sat Sun)
set ldp = (Seg Ter Qua Qui Sex Sab Dom)
set lmi = (Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec)
set lmp = (Jan Fev Mar Abr Mai Jun Jul Ago Set Out Nov Dec)
set d = `date`; indice $d[1] $ldi; set j = $ldp[$status]
indice $d[2] $lmi; set m = $lmp[$status]
echo $j $d[3] $m $d[6] ´   ´ $d[4]
```

Referências

Unix

[Gil92] D. Gilly. *U*nix in a nutshell: System V Edition. O'Reilly & Associate, segunda edição, Junho 1992. O'Reilly homepage: http://www.ora.com/.

[TSP93] G. Todino, J. Strang e J. Peek. *L*earning the Unix Operating System. O'Reilly, terceira edição, Agosto 1993. 108 páginas.

[webu1] http://www.hsrl.rutgers.edu/ug/unix_history.html A Brief history of Unix .

[webu2] http://milieu.grads.vt.edu/unix_history.html A Brief history of Unix .

Programação Unix

[KP84] B. Kernighan e R. Pike. *T*he Unix Programming Environment. Prentice-Hall, 1984. Mais uma referência do Kernighan em termos de programação em ambiente Unix.

[Rif93] J.M. Rifflet. *L*a programmation sous Unix. Ediscience, terceira edição, 1993. Excelente e completíssimo livro para quem lê francês.

Linux

[WK96] M. Welsh e L. Kaufman. *R*unning Linux. O'Reilly, segunda edição, 1996. muito bom livro para quem quer instalar e usar o Linux.

[ftp] ftp://sunsite.unc.edu/pub/Linux/docs/linux-doc-project/install-guide/. *I*nstallation and Getting Started Guide. M. Welsh e outros, Março 1998, versão 3.2. Guia de instalação e de introdução ao sistema Linux, disponível on-line.

[weba] http://gil.di.uminho.pt/. Grupo de investigação linux homepage. Página dos representantes portugueses do Linux Documentation Project.

[webb] http://sunsite.unc.edu/LDP/. Linux documentation project homepage. Fonte enorme de informação e de links para o mundo Linux (documentos on-line, distribuidoras, etc...).

[webc] http://www.linux.org/. Linux homepage. Mais uma fonte enorme de informação e de links para o mundo Linux (documentos on-line, distribuidoras, etc...).