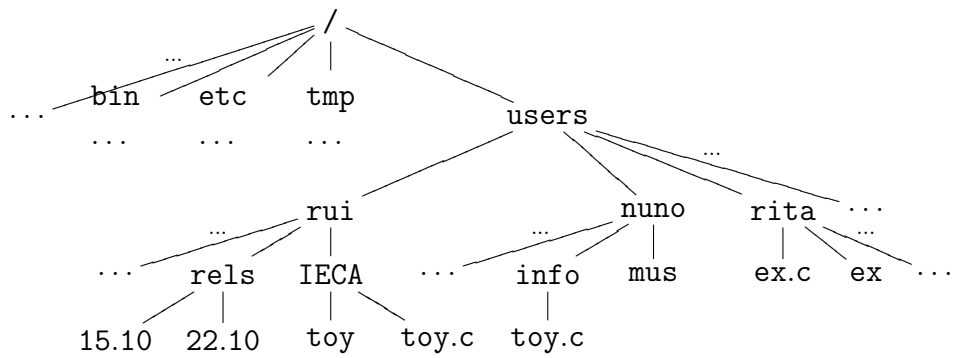
Breve introdução ao UNIX, EMACS, e GDB

Informática (Eng. C. Agrárias) – 1997/98

Ana Paula Tomás
Dep. Ciência de Computadores, FCUP

1 Alguns comandos UNIX

Os sistemas UNIX usam uma estrutura em árvore para organizar os *directórios* (i.e., colecções de ficheiros ou directórios) e ficheiros. Cada nó interno na árvore é um directório, e cada folha da árvore pode ser um ficheiro ou um directório vazio. Considere por exemplo,



`/` é um directório, o qual se diz *raíz* da árvore.

`/users`, `/users/rui`, `/users/rui/IECA` são directórios.

`/users/rui/IECA/toy.c` é (de esperar que seja) um ficheiro.

Cada ficheiro ou directório é identificado pelo caminho desde a raíz até ao ficheiro, ou dando o caminho relativamente a outro directório.

Se o *directório corrente* (i.e., onde está a trabalhar; pode ver qual é dando o comando `pwd`) for `/users/rui` então

`IECA`	designa `/users/rui/IECA`
`IECA/toy`	designa `/users/rui/IECA/toy`
`.`	designa o directório corrente, ou seja `/users/rui`
`..`	designa o *pai* do directório corrente, ou seja `/users`
`../rita`	equivale `/users/rita`
`../../etc`	equivale `/etc`
∼	*directório casa* (i.e., o corrente quando entra no sistema)
∼*nome*	directório casa do utilizador cujo "login" é *nome*

1.1 O Manual

man - para ver informação sobre comandos ou funções. Exemplo: `man ls` apresenta páginas de informação sobre comando `ls`. Para sair, carregar em "q".

1.2 Informação sobre ficheiros

ls - ("**list**") para ver quais os ficheiros num directório, ou informação sobre ficheiros. Se a identificação dada não estiver correcta (por exemplo, se não corresponder a qualquer ficheiro no directório dado, ou se não tiver permissão para aceder ao ficheiro) é indicado um erro.

Suponha que o directório corrente é `/users/nuno`. Dando,

`ls`	resultou	`info mus`
`ls ../rui/IECA`	resulta	`toy toy.c`
`ls info`	resulta	`toy.c`
`ls ../rui/IECA/toy`	resulta	`toy`

Utilização:

`ls` *directórios ou ficheiros*
`ls` *opções directórios ou ficheiros*

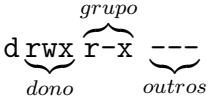
```
ls -l  /users/rui   /users/nuno  toy.c
ls -a  /users/nuno
ls -l
```

ls -l para ver as permissões, o dono, o grupo a que pertence o dono, o tamanho, a data de última modificação, e o nome dos ficheiros num directório

Exemplo dum resultado após o comando `ls -l`

```
drwxr-x---   2 ajt   prof  1024 Jun 14  1996 pesquisa/
drwxr-xr-x   2 ajt   prof  1024 Oct 24 18:35 public/
drwx------   6 ajt   prof  1024 May 16  1994 new/
-rw-r-----   1 ajt   prof  1551 Mar  5  1997 mudar
-rwxr-x---   1 ajt   prof  6890 Nov 19 10:24 teste*
-rw-r-----   1 ajt   prof   176 Nov 19 10:24 teste.c
-rw-r-----   1 ajt   prof   176 Nov 19 10:22 teste.c~
```

`d` - directório, `r` ("read") - permissão de leitura/cópia, `w` ("write") - permissão de escrita/alteração, `x` ("execute") - permissão de executar (no caso de directórios significa que pode explorar/descer o directório). Quanto às permissões:



`ajt` - o *dono*; `prof` - *grupo* de utilizadores a que pertence `ajt`; os *outros* são os restantes utilizadores.

ls -a mostra todos os ficheiros num directório, incluindo aqueles cujo nome começa por `.`, os quais normalmente não aparecem

1.3 Copiar ou Deslocar ficheiros

cp - ("**copy**") para copiar ficheiros

Suponha que o directório corrente é `/users/rui/IECA`.

```
cp  toy.c  ../rels
```

copia para o ficheiro `toy.c` que está em `/users/rui/rels`, o conteúdo do ficheiro `/users/rui/IECA/toy.c` (se não existir um ficheiro `toy.c` em `/users/rui/rels`, é criado);

```
cp  toy.c  outro
```

copia `toy.c` para o ficheiro `outro` que está no mesmo directório (se `outro` não existir, é criado um ficheiro com esse nome);

```
cp  toy.c  toy  ../rels
```

os ficheiros `toy.c` e `toy` são copiados para o directório `../rels`. É idêntico a fazer

```
cp  toy.c  ../rels
cp  toy    ../rels
```

Utilização:

`cp`	$nome_1$	$nome_2$
`cp`	$nome_1 \ldots nome_k$	*directório*

onde $nome_i$ identifica um ficheiro.

mv - ("**move files**") para mudar ficheiros dum directório para outro, ou para mudar o nome dum ficheiro (CUIDADO! verificar se o novo nome identifica um ficheiro já existente, cujo conteúdo nos interessa, o qual seria destruído).

Utilização:

`mv`	$nome_1$	$nome_2$
`mv`	$nome_1 \ldots nome_k$	*directório*

Suponha que o directório corrente é `/users/rui/IECA`.

```
mv  toy.c  ../rels
```

muda `toy.c` para `/users/rui/rels`. Se já existisse um ficheiro `toy.c` em `/users/rui/rels`, o seu conteúdo seria substituído pelo novo;

```
mv  toy.c  outro
```

muda o nome de `toy.c` passando a ser `outro`;

```
mv  toy.c  toy  ../rels
```

muda os dois ficheiros para o directório `/users/rui/rels`.

1.4 Criar e Remover directórios

mkdir - ("**make directory**") criar directórios

Utilização:

`mkdir`	$nome_1$
`mkdir`	$nome_1 \ldots nome_k$

para criar um directório cuja identifcação é $nome_1$, ou vários, designados por $nome_1$, ..., $nome_k$.

Exemplo:

```
mkdir  /users/rita/informatica
mkdir  /users/rita/programas  /users/rita/relatorios
```

Mas, se o directório corrente for `/users/rita` basta fazer

```
mkdir  informatica
mkdir  programas  relatorios
```

rmdir - ("**remove directory**") remover directórios (que se encontram vazios)

Utilização:

`rmdir`	$nome_1$
`rmdir`	$nome_1 \ldots nome_k$

1.5 Mudar de directório

cd - ("**change directory**") alterar directório corrente para passar a ser o directório indicado

Utilização:

`cd`	*nome*

Se o directório corrente for `/users/rita` e quiser que passe a ser `/users/rui/IECA`, dar um dos comandos:

```
cd  /users/rui/IECA
cd  ../rui/IECA
```

`cd` (sem argumentos) equivale a `cd` $\sim$, pelo qual vai para o directório casa.

1.6 Remover ficheiros

rm - ("**remove files**") para remover ficheiros que já não são necessários.

Utilização:

`rm`	*nome*
`rm`	$nome_1 \ldots nome_k$
`rm -i`	$nome_1 \ldots nome_k$

Com a opção `-i`, a execução é interactiva, perguntando ao utilizador se quer mesmo remover o ficheiro que tem aquele nome. CUIDADO! remover um ficheiro significa "deitá-lo ao lixo", por isso se quiser duplicar a segurança, use a opção `-i`.

1.7 Ver conteúdo de ficheiros

`cat` - mostra conteúdo dum ficheiro (duma só vez).

`more` - mostra conteúdo página a página. Para passar à linha seguinte carregar na tecla "enter", para ver a próxima página carregar na barra de "espaço", e para sair em "q".

`less` - semelhante a `more` mas permite andar para trás e para a frente no ficheiro.

Utilização:

`cat`	*nome*
`more`	*nome*
`less`	*nome*

1.8 Edição de ficheiros

O editor de texto que usamos é o `emacs`. Para o lançar, dar o comando

```
emacs &
```

CUIDADO! Se se esquecer de escrever &, é melhor **sair** do `emacs` (ver abaixo como fazer isso), e voltar a entrar.

Dentro do `emacs` pode dar comandos ao `emacs`, selecionando (com o botão esquerdo do rato) os comandos nos menus `Buffers Files Tools Edit Search ...Help`.
Também pode dar os comandos a partir do teclado.
Se abrir o menu `Files` encontra entre outras opções, as seguintes.

`Open File ...`	`(C-x C-f)`	para abrir um ficheiro, cujo nome vai indicar no fundo da janela do `emacs`.
`Save Buffer...`	`(C-x C-s)`	para guardar o ficheiro, após alterações.
`Exit Emacs`	`(C-x C-c)`	para sair do `emacs`.

Se usar o teclado, para abrir um ficheiro pode fazer `(C-x C-f)`, o que quer dizer carregar *simultaneamente* nas teclas Ctrl e X, e depois Ctrl e F.

No menu `Edit` encontra, entre outras, as seguintes opções.

`Undo`	`(C-_)`	para voltar ao que tinha antes duma alteração.
`Cut`	`(C-w)`	para cortar uma região *seleccionada* (pressionando o botão esquerdo do rato, passe na região que quer seleccionar)
`Copy`		para copiar uma região seleccionada.
`Paste Most Recent`	`(C-y)`	para colocar na posição onde está o cursor, o que acabou de copiar ou cortar; pode também carregar no botão central do rato.

Para alterar a posição do cursor pode usar o rato — escolha a nova posição, e carregue no botão esquerdo do rato — ou pode usar, por exemplo, as teclas ←, →, ↑, e ↓.

Se estiver a dar comandos a partir do teclado, e por engano der `(C-s)` estando selecionada a janela da `shell` (i.e., onde pode dar comandos UNIX) em vez da janela do `emacs`, **fica sem poder dar comandos UNIX**. Para voltar a poder, fazer `(C-q)`.

Se **algo correr mal** ao dar comandos ao `emacs`, experimente fazer `C-g`, ou em último caso, sair do `emacs` (CUIDADO! porque nesse caso, pode perder as últimas alterações que fez). **Se estiver a fazer muitas alterações num ficheiro, de tempos a tempos salve-o!** (`C-x C-s`)
Quando altera um ficheiro já existente, o `emacs` cria uma cópia do anterior, que fica com um nome idêntico mas terminando em `~` (por exemplo, `teste.c~`).

No menu `Help` tem `Emacs Tutorial (C-h t)` que lhe permite obter mais informações — **tutorial sobre o `emacs`**. Se fizer só `C-h` também pode obter ajuda.

1.8.1 Edição de programas em linguagem C

Se abrir um ficheiro cujo nome tem extensão `.c` (ou seja, termina em `.c`), o `emacs` entra em modo de edição para programas em C, aparecendo (C) na barra situada na parte inferior da janela do `emacs`.
Escreva num ficheiro (por exemplo, `exemplo.c`) o seguinte programa.

```
#include <stdio.h>

void main()
{
int x;
scanf("%d",&x);
printf("o numero que escreveu: %d",x);
}
```

Se o `emacs` estiver em modo de edição (C), coloque o cursor no topo do ficheiro (ou na instrução `#include <stdio.h>`, e carregue na tecla "Tab"

e proceda, de modo análogo, para as restantes linhas. Caso não obtenha

```
#include <stdio.h>

void main()
{
  int x;
  scanf("%d",&x);
  printf("o numero que escreveu: %d",x);
}
```

verifique, por exemplo, se se esqueceu de algum "`;`", ou de alguma chaveta.

1.9 Compilar e executar programas em linguagem C

Depois de editar o texto do programa em C, e o guardar num ficheiro cujo nome termine em `.c`, seja por exemplo, `teste.c`, para obter o *executável* correspondente ao programa em `teste.c` dar o comando (na "shell")

```
gcc  teste.c
```

o qual, **se não ocorrerem erros**, cria no directório corrente (onde deve também estar `teste.c`), o ficheiro `a.out` que é o *executável* correspondente ao programa.

Para **executar** o programa dar o comando

```
a.out
```

Se quiser atribuir um nome ao executável, diferente do nome `a.out` atribuído por defeito, deve dar o comando

`gcc -o` *nome* `teste.c`

onde *nome* é a designação que pretende dar. Assim,

```
gcc -o teste teste.c
```

cria ou substitui o executável `teste` no directório corrente. Neste caso, para executar o programa dar o comando

```
teste
```

Se **o programa não funcionar** como esperamos (por exemplo, se **não parar**), ou se o quiser fazer parar, carregue simultaneamente nas teclas Ctrl e C.

Se ocorrerem **erros na compilação**, veja o relatório de erros que saiu, e tente corrigi-los: volte a editar `teste.c` para fazer as correcções (não se esqueça de o guardar depois das alterações), e tente novamente compilar o programa. A falta dum só "`;`" ou a não declaração (ou declaração incorrecta) duma variável, podem dar origem a muitos erros de compilação, não sendo dito que falta o "`;`".

Utilização:

`gcc`	*nome*`.c`	
`gcc`	`-o` *nomeexec*	*nome*`.c`
`gcc`	`-g -o` *nomeexec*	*nome*`.c`
`a.out`		
nomeexec		

1.9.1 Executar o programa passo a passo

Se o **programa não funcionar** como queria, pode voltar a analisar com cuidado o programa que escreveu em C, ou pode recorrer a um *debugger* — programa de ajuda na correcção de programas.
Para isso, deve voltar a compilar o programa, mas com a opção `-g`, ou seja, dar o comando

```
gcc -g -o teste teste.c
```

ou em geral

`gcc -g -o` *nomeexec nome*`.c`

e a seguir dar o comando

`gdb teste`

ou em geral

`gdb` *nomeexec*

o qual chama o *debugger*. Agora, no `gdb` pode ver uma simulação da execução do programa `teste` (ou, de *nomeexec*) **passo a passo**.

Para isso, dentro do `gdb` dê o comando

`break main`

depois, `run`, e finalmente `step` sucessivamente, até o programa terminar.

Se quiser ver o valor que uma variável tem num dado passo, dê o comando

`print` *nome*

em que *nome* deve ser substituído pelo nome que deu, em `teste.c`, à variável que quer inspecionar.

Se não quiser parar em todos os passos, pode colocar **pontos de paragem** ("breakpoints"), em subprogramas ou em instruções, dando ao `gdb` comandos

`break` *nome*

para parar quando entrar no subprograma cujo nome é *nome*, ou

`break` *número*

para parar quando chegar à instrução que no ficheiro `teste.c` está na linha cujo número é *número*. Para ver o número da linha em que está a instrução em que quer parar, pode, no `emacs`, posicionar o cursor na linha de `teste.c` pretendida. Na barra que está na parte inferior da janela do emacs aparece **L***número*.

Como anteriormente, se depois de colocar "breakpoints", quiser executar o programa dê o comando

`run`

O programa é executado até chegar a um ponto de paragem. Nessa altura pode

- inspecionar os valores das variáveis do programa: só deve ter acesso às variáveis globais (se as houver), e às locais ao subprograma ou função que está a ser executado.
- continuar até ao próximo *breakpoint*: dê o comando `cont` ("continue").
- prosseguir *passo a passo*: dê o comando `step`, ou `next` (este último permite passar para a próxima linha no programa `teste.c`);
- colocar novos breakpoints;
- apagar breakpoints:
  - `delete` para apagar todos os breakpoints colocados;
  - `delete` *número* para apagar o breakpoint cujo número é *número*. O número atribuído a um breakpoint é indicado pelo `gdb` imediatamente após colocar esse breakpoint, ou quando pára nesse breakpoint;
- dar o comando `quit` para sair do `gdb`; Ctrl C para interromper execução do programa `teste` (se este não parar).
- dar o comando `help` para **obter informação sobre os comandos** possíveis.

Como exemplo, siga a execução do programa seguinte no `gdb` (antes dar `step`, veja o valor de `x`, fazendo `print x`).

```
#include <stdio.h>
void main()
{
  int x;

  x = 7;
  if (x > 2) printf("\n%d maior do que 2\n",x);
  while (x < 15) {
    printf("no ciclo \"while\"  x = %d\n",x);
    x += 2;
  }
  for(x=7; x < 15; x += 2)
    printf("no ciclo \"for\"  x = %d\n",x);
  do {
    x = x + 4;
    printf("no ciclo \"do\"  x = %d\n",x);
  } while (x < 29);
}
```

1.10 Sumariando...

`ls`		"list"
`ls -l`		
`ls`	*nome*$_1$ ... *nome*$_k$	
`ls -l`	*nome*$_1$ ... *nome*$_k$	
`cp`	*fich*$_1$ *fich*$_2$	"copy"
`cp`	*fich*$_1$ ... *fich*$_k$ *dir*	
`mv`	*fich*$_1$ *fich*$_2$	"move"
`mv`	*fich*$_1$ ... *fich*$_k$ *dir*	
`mkdir`	*dir*$_1$ ... *dir*$_k$	"make directories"
`rmdir`	*dir*$_1$ ... *dir*$_k$	"remove directories"
`rm -i`	*fich*$_1$ ... *fich*$_k$	"remove files"
`cd`	*dir*	"change directory"

no emacs

`C-x C-f`	"open file"
`C-x C-s`	"save"
`C-g`	
`C-w`	"cut"
`C-y`	"paste"
`C-_`	"undo"
`C-x C-c`	"quit"

```
emacs &
```

no gdb

`break`	*nome*
`break`	*line*
`delete`	*breakpoint*
`delete`	
`run`	
`cont`	
`step`	
`next`	
`print`	*var*
`quit`	
`help`	
`C-c`	

`gcc`		*nome*`.c`	
`gcc`	`-o`	*nomeexec*	*nome*`.c`
`gcc`	`-g -o`	*nomeexec*	*nome*`.c`
`gdb`		*nomeexec*	