11.6 O Utilitário Unix. *make* o Gestor do Projectos

Introdução:

O Utilitário *make* é uma ferramenta do sistema operativo Unix. É um gerador de comandos que é muito utilizado como uma ferramenta de *engenharia de software* para a compilação, gestão e manutenção de programas. O *make* funcione através dum ficheiro de **descrições** que gere comandos executados pelo Shell de Unix. Os comandos mais utilizados tem a ver com a gestão e manutenção dum projecto de Software - a criação dos ficheiros intermediários, linkagem, criação dum ficheiro executável, removação de ficheiros temporários, instalação, criação de relatórios etc.

make é muito utilizado para gerir as *dependências* entre ficheiros. Qualquer projecto de software envolve vários ficheiros que tem dependências entre si. Por exemplo um executável tem que ser criado através dum processo de *linkagem* de vários ficheiros objectos e livrarias, que são dependentes em vários ficheiros de código fonte. Alterando um ficheiro implique a recompilação de alguns, mas nem todos, dos ficheiros do projecto. O *make* visa gerir este tipo de procedimento. Hoje em dia o desenvolvimento de Software é um processo de equipa. Make ajude manter as relações entre os ficheiros dum projecto complexo coordenando as actividades da equipa.

Um Ficheiro Simples de Make

Por convenção o ficheiro de descrição tem nome **Makefile** e o utilitário make, chamado do shell de Unix, leia esta ficheiro. Por exemplo
> make program
indique que queremos criar a mais nova versão, do *"program"* e todos os passos necessário de compilação e linking necessários parar criar *program* são descritos no ficheiro Makefile

Sintaxe dum Ficheiro Makefile

- linhas de comentários: texto que comece com o cardinal (#)
- linhas de definição de variáveis : var = valor.
  Referências às variáveis são feitas com o símbolo $(var)
- linhas de Alvo (target)
  O *Alvo* é o que nos pretendemos criar. Isto é separado pelos suas dependências através dum ":" Nas linhas seguintes são os regras/comandos. Este linhas começam obrigatoriamente com **um tab**. Portanto os comandos especifiquem como é que o alvo é criado a partir das suas dependências.

Exemplo 1

```
#Makefile para meu projecto
CC=gcc
projecto : main.c ordenacao.o
        $(CC) –o projecto main.c ordenacao.o
ordenacao.o : ordenacao.c
        $(CC) –c ordenacao.c
```

Neste exemplo a variável CC especifica o compilador definido como o gcc. A seguir há dois alvos ordenacao.o e projecto. Repare que o alvo projecto depende na criação do ficheiro ordenacao.o. Na linha de comando chamamos o utilitário make usando

> make ordenacao
> make projecto

A maneira mais simples de utilizar make é de simplesmente escrever "make" que tentará criar o primeiro alvo. O utilitário depois verificará automaticamente quais são os ficheiro(s) que precisam de ser criados.

Exemplo 2

Suponha que estamos a escrever um programa para uma lista-ligada com nome do executável **lista**. O programa tem:

- 3 ficheiros de código fonte c → main.c iointerface.c listafn.c
- um ficheiro de assembler para rotinas de ordenação → sort.s
- um conjunto de rotinas numa livraria estática → /usr/projecto/biglib.a

Para criar o executável a sequência de comandos seria

```
> cc -c main.c            > cc -c iointerface.c
> cc -c listafn.c         > as -o sort.o sort.s
> cc -o lista main.o iointerface.o listafn.o sort.o /usr/projecto/biglib.a
```

Claro que todos estes ficheiro possam ser compilado e linked num único comando de compilação. No entanto dentro dum projecto grande e complexo isto não seria viável. Cada código fonte pode ser escrito, compilado e até testado separadamente do resto do projecto e uma aplicação pode demorar muito tempo para ser compilado, portanto senso comum sugere que se aproveita compilações previas.

Agora vamos ver como é que isto pode ser especificado usando um ficheiro de descrições chamado Makefile

```
# Makefile   para listaligada
# O compilador de C
CC = cc
# O compilador de Assembly
AS = as

lista : main.o iointerface.o  listafn.o sort.o o   /usr/projecto/biglib.a
  $(CC) -o lista main.o  iointerface.o  listafn.o  sort.o   /usr/projecto/biglib.a

main.o : main.c
  $(CC) -c main.c

iointerface.o : iointerface.c
  $(CC) -c iointerface.c

listafn.o : listafn.c
  $(CC) -c listafn.c

sort.o : sort.s
   $(AS) -o sort.o sort.s

# limpar e remover ficheiros desnecessários
limpar :
   /bin/rm -f   core  *.o
```

- O alvo *main.o* é criado a partir do ficheiro main.c que é compilado com o comando cc -c main.c
  Para criar este alvo > **make main.o**

- O alvo *limpar* não tem dependências, simplesmente execute o comando rm -f que vai remover os ficheiro core e objecto.
  Para criar este alvo > **make limpar**

- O alvo *lista* depende em cinco ficheiros que quando existem são linkados para criar o executável
  Para criar este alvo > **make lista**

Verificação das Dependências

A utilidade de *make* reside aqui. Considere este exemplo quando desejamos criar o executável lista através do comando *make lista* e um dos ficheiros objecto não existe ou que um dos seus ficheiros de código fonte tenha sido alterado e portanto um dos ficheiros objecto refere uma versão velha ! *Make* a ser invocado primeiro verifique as dependências do alvo e execute um mínimo número de comandos!

i) se um dos ficheiros das dependências não existe procure para ver se um dois outros alvos especifica como é este ficheiro é criado.

ii) se uma das dependências dum alvo é mais nova do que a versão actual do alvo então o alvo será recriado. (nota que o sistema operativo guarde o ultimo tempo de modificação dum ficheiro e o tempo da sua criação)

iii) repete este processo recursivamente !

CASO 1: todos os ficheiro fonte alterados

```
ciunix> make lista
cc -c main.c
cc -c iointerface.c
cc -c listafn.c
as -o sort.o sort.c
cc  -o lista main.o  iointerface.o  listafn.o sort.o /usr/projecto/biglib.a
```

CASO 2: apenas o ficheiro main.c alterado

```
ciunix> make lista
cc -c main.c
```

CASO 3 : novo ficheiro de livraria biglib.a

```
ciunix> make lista
$(CC) -o lista main.o  iointerface.o  listafn.o  sort.o   /usr/projecto/biglib.a
```

CASO 4 : nenhum ficheiro alterado .. não vale a pena recompilar !

```
ciunix> make lista
'lista' is up to date
```