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 gestão e manutenção de programas.

Make funcione através um ficheiro de descrições que gere comandos para serem executados pelo Shell de Unix. Os comandos que vamos descrever. e que são mais utilizados. tem a ver com a gestão e manutenção dum projecto de Software. Incluindo nesta definição são a criação dos ficheiros intermediários, a criação dum ficheiro executável, removação de ficheiros temporários, instalação *linking,* 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 um processo de *link* 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. Hoje em dia o desenvolvimento de Software é um processo de equipa. Make ajude manter as relações entre os ficheiros dum projecto complexo coordenado as actividades da equipa.

Um Ficheiro Simples de Make

Por convenção o ficheiro de descrição está dentro do ficheiro com nome Makefile e o utilitário make, chamado do shell de Unix, leia esta ficheiro. Por exemplo

> make program

Indique que queremos criara uma versão, a mais nova, do *"program"*. Portanto se *program* for um executável o comando indique que queremos fazer todos os passos necessário de compilação e linking necessários parar criar *program*.

Suponha que estamos a criar um executável, lista, um programa para uma lista-ligada que 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 → /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 tudo possa ser compilado e linked num único comando de cc. No entanto dentro dum projecto complexo isto não é 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 com um ficheiro de descrições chamado Makefile (ver ~crocker/crocker/prog3/listaligada

```
# Makefile   para listaligada
# O compilador de C , podia ser cc ,  gcc etc.
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
```

Este ficheiro contem

- 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) - o sintaxe é parecido com o Shell.

- linhas de Alvo (target). Nome do alvo e subsequente linhas especificando as dependências, regras e comandos para executar.

O Alvo é o que nos pretendemos criar. Isto é separado pelos suas dependências através um colon :. 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.

Por exemplo

- 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 juntados com o linker via o comando
  cc -o lista main.o iointerface.o listafn.o sort.o /usr/projecto/biglib.a

  Para criar este alvo **> make lista**

Verificação das Dependências

A utilidade de *make* reside aqui. Considere o caso em cima quando desejamos criar o executável lista através do comando *make lista* e um dos ficheiros objecto não existe ou que um dos seus ficheiro 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 !

i) se um dos ficheiros dos dependências não existe procure no resto das suas linhas para uma indicação como é que este ficheiro necessário pode ser criado.

ii) Para qualquer dependência verifique para ver se alguma dos dependências é mais nova do que a versão velha *lista*. Isto é fácil como Unix guarde sempre o ultimo tempo de modificação dum ficheiro.

iii) repete um processo recursivo para cada dependência, verificando cada dependência quando puder.

Make execute um mínimo número de comandos.. ver os exemplos a seguir

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
cc  -o lista main.o  iointerface.o  listafn.o
```

CASO 3 : ficheiro main.c alterado mas criado o ficheiro objecto)

```
ciunix> make lista
cc  -o lista main.o  iointerface.o  listafn.o
```

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

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