

Programar em C/Makefiles

De Wikibooks

< Programar em C

Tabela de conteúdo

- 1 Makefile
 - 1.1 Sintaxe de criação do arquivo
 - 1.1.1 Regras complementarias:
 - 1.1.2 Definir Variáveis:
 - 1.1.3 Variáveis Personalizadas:
 - 1.1.4 Variáveis internas:
 - 1.1.5 As regras de interferência:
 - 1.1.6 Sub Makefiles:
 - 1.1.7 Make install:

Makefile

O makefile é um arquivo utilizado por um programa chamado make ou em outros sistemas operativos ,gmake, nmake, tmake, etc.

Ele pode variar de um sistema a outro pois não faz parte de nem uma normalização .

O texto contido em um makefile e usado para a compilação de um projeto. Vantagens do uso.

Evita a compilação de arquivos desnecessários.

Por exemplo, se seu programa utiliza 120 bibliotecas e você altera apenas uma, o makefile descobre (comparando as datas de alteração dos arquivos fontes com as dos arquivos anteriormente compilados) qual arquivo foi alterado e compila apenas a biblioteca necessária.

As explicações a seguir são para o utilitário GNU make.

Então vamos para a apresentação do Makefile através da compilação de um pequeno projeto em linguagem C.

- Criar uma pasta com esses 4 arquivos :

teste.c ,teste.h , main.c, Makefile.

- De um nome para a pasta Projeto.

```

/*===== teste.c =====*/
#include <stdio.h>
#include <stdlib.h>
/*Uma função makeTeste()*/
void makeTeste(void){
    printf("O Makefile e super Legal\n");
}

```

Aqui escrevemos o header :

```

/*===== teste.h =====*/
/*===== Cabeceira ou header =====*/
#ifndef _H_TESTE
#define _H_TESTE
/*A nossa função
void makeTeste(void);
/* De um enter depois de endif*/
/*Para evitar warning*/
#endif

```

Agora a função main :

```

/*===== main.c =====*/
#include <stdio.h>
#include <stdlib.h>
#include "teste.h"
/* Aqui main ;( */
int main(void){
    makeTeste();
    return (0);
/*tkx Everton*/
}

```

Para compilar fazemos um arquivo Makefile minimal.

```

#Para escrever comentários ##
##### Makefile #####
all: teste
teste: teste.o main.o
    gcc -o teste teste.o main.o
#----> Distancia com o botao TAB ### e nao com espaços
teste.o: teste.c
    gcc -o teste.o -c teste.c -W -Wall -ansi -pedantic
main.o: main.c teste.h
    gcc -o main.o -c main.c -W -Wall -ansi -pedantic
clean:
    rm -rf *.o
mrproper: clean
    rm -rf teste

```

Para não ter erros os espaços devem ser feito com a tecla TAB.

E compilar é só ir dentro da pasta "Projeto" apertar F4 escrever make e apertar enter.

Uma vez compilado podemos modificar teste.c . Se teste.c foi modificado então make modifica teste.o e se não deixa teste.o como esta.

- all : É o nome do arquivo binário que sera criado "Nome do programa"
- teste: teste.c .Pode ser interpretado com arquivo_de_destino: arquivo_de_origem.
- clean: Apaga os arquivos intermediários.Se você escrever no console make clean

ele apaga os arquivos objeto da pasta.

- mrproper: Apaga tudo o que deve ser modificado.No console escreva make mrproper

Sintaxe de criação do arquivo

O makefile funciona de acordo com regras, a sintaxe de uma regra é:

```
regra: dependências
Apertar o botão TAB  comando
                      comando ...
```

Regras complementarias:

- all : É o nome do arquivo binário que sera criado "Nome do programa"
- clean: Apaga os arquivos intermediários.
- mrproper: Apaga tudo o que deve ser modificado.

Definir Variáveis:

As variáveis servem para facilitar o trabalho.

Em vez de mudar varias linhas mudamos só o valor da variável.

Deve ser por isso que se chama variável não?

Definimos da forma seguinte.

```
NOME=VALOR
E para utilizar esta variável colocamos entre $() .
Então ela vai ficar assim $(NOME)
```

Vamos para o exemplo com o nosso Makefile.

Colocamos em vez de :

- NOME SRC
- E em vez de VALOR main.c .
- E para poder usar \$(SRC)

Será que na pratica funciona?? Vamos ver..

```

#Para escrever comentários ##
##### Makefile #####
#Definimos a variável
SRC=main.c
all: teste
teste: teste.o main.o
    gcc -o teste teste.o main.o
#----> Distancia com o botao TAB ### e nao com espaços
teste.o: teste.c
    gcc -o teste.o -c teste.c -W -Wall -ansi -pedantic
#
#Coloquei $(SRC) em todos os lugares aonde estava main.c
main.o: $(SRC) teste.h
    gcc -o main.o -c $(SRC) -W -Wall -ansi -pedantic
clean:
    rm -rf *.o
mrproper: clean
    rm -rf teste

```

Todos os lugares do código que contem o CONTEÚDO da variável e modificado colocando em seu lugar o NOME da variável.

Variáveis Personalizadas:

- CC=gcc .Definimos CC para nomes de compiladores de C ou C++ .Aqui o gcc.
- CFLAGS=-W -Wall -ansi -pedantic .Serve para definir opções passadas ao compilador.

Para o c++ o NOME e CXXFLAGS .

- LDFLAGS e utilizado para editar as opções de links.
- EXEC=teste .EXEC define o NOME do futuro programa executável.
- OBJ=teste.o main.o . Para cada arquivo.c um arquivo OBJETO e criado com a extensão ".o" arquivo.o .

Então e só olhar na sua pasta todos os arquivos com a extensão ".c" e colocar na variável OBJ com a extensão".o" .

- Outra maneira e mesma coisa. OBJ agora e igual a main.o teste.o

```

SRC = main.c teste.c
OBJ= $(SRC:.c=.o)

```

- E super manero a tua idéia camarada.
 - Mais tenho 200 arquivos.c e não quero olhar o nome de todos um por um.
 - Tem outra idéia??
 - Poderíamos utilizar *c mais não podemos utilizar este caracter joker na definição de uma variável.
 - Então vamos utilizar o comando " wildcard " ele permite a utilização de caracteres joker na definição de variáveis.
- Fica assim.

```
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)
```

- Observação se quiser fazer aparecer uma mensagem durante a compilação escreva @echo "Minha mensagem" .
- E mais tem um monte de mensagens e fica muito feio
- Tem outra idéia??.. O pessoal vamos parando ;) não sou uma maquina de idéias.
- Para deixar as mensagens em modo silencioso coloque "@" no começo do comando.
- Fica assim

```
@$(CC) -o $@ $^
```

Variáveis internas:

```
$@ Nome da regra.
$< Nome da primeira dependência
$^ Lista de dependências
$? Lista de dependências mais recentes que a regra.
$* Nome do arquivo sem sufixo
```

As regras de interferência:

Não disse nada antes porque estávamos no estado principiantes "noob". São regras genéricas chamadas por default.

- .c.o : .Ela significa fazer um arquivo.o a partir de um arquivo.c .
- %.o: %.c .A mesma coisa. A linha teste.o: teste.c pode ser modificada com essa regra.
- .PHONY: .Preste bem atenção. Esta regra permite de evitar conflitos.
 - Por exemplo "clean:" e uma regra sem nem uma dependência não temos nada na pasta que se chame clean.
 - Agora vamos colocar na pasta um arquivo chamado clean. Se você

tentar apagar os "arquivos.o" escrevendo "make clean" não vai acontecer nada porque make diz que clean não foi modificado.

- Para evitar esse problema usamos a regra .PHONY : . Fica assim.
- .PHONY: clean mrproper
- .PHONY: diz que clean e mrproper devem ser executados mesmo se arquivos com esses nomes existem.

Agora vamos modificar mais uma vez o nosso Makefile com tudo o que sabemos sobre variáveis.

```

#Para escrever comentários ##
##### Makefile #####
#Definimos a variável
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
EXEC=teste
OBJ=teste.o main.o
all: $(EXEC)
@echo "Vou comer a compilação"
#Não coloquei a variável OBJ para que possam entender as variáveis internas.
#Se entenderão podem colocar $(OBJ) no lugar de teste.o main.o
teste: teste.o main.o
# $@ = teste:
# $^ = teste.o main.o
# $(CC) -o $@ $^
# teste.o:teste.c
%.o: %.c
# $(CC) -o $@ -c $< $(CFLAGS)
main.o: main.c teste.h
# $(CC) -o $@ -c $< $(CFLAGS)
.PHONY: clean mrproper
clean:
# rm -rf *.o
@echo "Compilação prontinha"
mrproper: clean
# rm -rf $(EXEC)

```

- Po legal ;) parece até trabalho de gente grande.

Sub Makefiles:

Ler tudo isso só para compilar um programa??

O sub-makefile é lançado por meio de um "Makefile principal" vamos simplificar para o Padrão Makefile.

Aonde estávamos??...A sim para que serve??

O Makefile Principal executa os sub-makesfiles de outras pastas.

Como ele faz??

```

Usamos uma variável pre-definida $(MAKE).

```

Bom ao trabalho.. Crie dentro da pasta "Projetos" outra pasta com o nome

"sub-make".Dentro da pasta sub-make crie um arquivo Makefile e um arquivo submake.c

Dentro da pasta sub-make coloque este Makefile.

```
#####Pasta:sub-make ## Makefile #####
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
EXEC=teste2
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)
all: $(EXEC)
@echo "compilando sub-makefile"
@echo "sub-makefile compilado"
teste2: $(OBJ)
@$(CC) -o $@ $^
.PHONY: clean mrproper
clean:
@rm -rf *.o
mrproper: clean
@rm -rf $(EXEC)
```

Agora vamos escrever o arquivo subteste.c .

```
#include <stdio.h>
#include <stdlib.h>
/* Informacao
 * Nao utilizem este código para fazer um kernel
 */
int main(void)
{
printf("Sou o binario que esta em sub-make");
printf("Finalmente em fim vivo graças ao Patrao Makefiles ;)");
return (0);
}
```

Agora retorne na pasta "Projeto" vamos modificar o Makefile .
Vamos colocar a seguinte linha:

```
@cd sub-make && $(MAKE)
```

- Explicando: "@" silencioso "cd" para abrir a pasta sub-make "&&" e executar make "\$(MAKE)"
- Vamos fazer a mesma coisa para "clean:" e "mrproper:" então ao executar "make clean" no console ele vai executar o mesmo comando no sub-makefile.

```
##### 0 Makefile principal #####
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
EXEC=teste
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)
all: $(EXEC)
@echo "Compilando Projeto"
@echo "O patrao foi compilado"
#A linha que vai compilar sub-make
@cd sub-make && $(MAKE)
teste: $(OBJ)
@$(CC) -o $@ $^
%.o: %.c
@$(CC) -o $@ -c $< $(CFLAGS)
main.o: main.c teste.h
@$(CC) -o $@ -c $< $(CFLAGS)
.PHONY: clean mrproper
clean:
@rm -rf *.o *~
# E a mesma coisa que dar um F4 dentro da pasta sub-make
# e escrever make clean
@cd sub-make && $(MAKE) $@
mrproper: clean
@rm -rf $(EXEC)
#modificamos aqui tambem
@cd sub-make && $(MAKE) $@
```

Não esqueça de dar TAB em todas as linhas que estão em baixo dos ":" dois pontinhos. OK agora e so dar um F4 dentro da pasta projetos e você tem treis comandos a disposição.

- make
- make clean
- make mrproper

Make install:

Automatisando a instalação do programa com a regra install: .

- install: .Coloca o binario ou executavel em uma determinada pasta ,/bin ou /usr/bin no linux

mais pode ser em qualquer outra.E so usar o comando "mv" ou "cp" mover ou copiar.

- Crie uma pasta bin dentro de "Projetos". Devem saber que nao devem colocar nada inutil que venha da internete na pasta raiz do linux.
- Vamos fazer duas variaveis:
 - prefix=/caminho/ate onde/esta/ Projetos
 - bindir=\$(prefix)/bin .Igual a /caminho ate/Projeto/dentro de Projeto a pasta bin .
 - E adicionarmos a regra install:all com seus comandos.

Modificando o make principal.

```
##### 0 Makefile principal #####
#Valeu Everton
#Coloque o caminho ate Projeto aqui
prefix=/home/USER/Projeto
bindir=$(prefix)/bin
CC=gcc
CFLAGS=-W -Wall -ansi -pedantic
EXEC=teste
SRC= $(wildcard *.c)
OBJ= $(SRC:.c=.o)
all: $(EXEC)
@echo "Compilando Projeto"
@echo "O patrao foi compilado"
#A linha que vai compilar sub-make
@cd sub-make && $(MAKE)
teste: $(OBJ)
@$(CC) -o $@ $^
%.o: %.c
@$(CC) -o $@ -c $< $(CFLAGS)
main.o: main.c teste.h
@$(CC) -o $@ -c $< $(CFLAGS)
#Entao depois e so executar make e depois make install
install:all
@mv $(EXEC) $(bindir)/
.PHONY: clean mrproper
clean:
@rm -rf *.o *~
# E a mesma coisa que dar um F4 dentro da pasta sub-make
# e escrever make clean
@cd sub-make && $(MAKE) $@
mrproper: clean
@cd bin && rm -rf $(EXEC)
#modificamos aqui tambem
@cd sub-make && $(MAKE) $@
```

Então quando você digitar no console "make" depois "make install" ele vai colocar o binario que esta em "Projetos" dentro de "bin".

Se você quiser colocar o binario que esta na pasta "sub-make" na pasta "bin"

- Copiar e colar no makefile da "sub-make" as variaveis "prefix" e "bindir" e a regra install: com seu comando.
- E no "Makefile principal" em baixo de "install:" coloque esta linha @cd sub-make && \$(MAKE) \$@
- Aqui eu modifiquei o "mrproper" porque agora os binarios que devem ser apagados com "make mrproper" estão em "bin".
- Vou deixar voces modificarem o "mrproper" do "sub-makefile" como pessoas adultas e responsaveis ;) Valeu galera.

Os comandos no console são:

- make

- make install
- make clean
- make mrproper .Para apagar os binarios.

Obtido em "http://pt.wikibooks.org/wiki/Programar_em_C/Makefiles"

Categoria: Programar em C

- Esta página foi modificada pela última vez às 14h23min de 5 de dezembro de 2008.
- Conteúdo disponível sob GNU Free Documentation License.