makefiles

kevinluo

Contents

目录

1 案例

1.1 通用 makefile, 自动遍历子目录源文件，自动生成依赖。

一份通用 makefile, 自动遍历子目录源文件，自动生成依赖 Ubuntu 和 OSX

这份 makefile 可以将当前 makefile 所在文件夹以及所有子文件夹中的 cpp 文件打包成静态库/动态库/可执行文件. 自动生成所有依赖关系，修改任何文件都可以触发重新编译相应依赖的文件。

在 Ubuntu 和 OSX 系统测试通过。

```
SHELL = /bin/bash

AllDirs := $(shell ls -R | grep '^\./.*:$$' | awk '{gsub(":","");print}') .
Sources := $(foreach n,$(AllDirs) , $(wildcard $(n)/*.cpp))
Objs := $(patsubst %.cpp,%.o, $(Sources))
Deps := $(patsubst %.cpp,%.d, $(Sources))
StaticLib := libyy.a
DynamicLib := libyy.so
Bin := yy

#AllLibs : $(DynamicLib)
#AllLibs : $(StaticLib)
AllLibs : $(Bin)

CC = g++
CXXFLAGS = -g -O2 -fPIC -Wall
CPPFLAGS = $(foreach n,$(AllDirs) , -I$(n))
LDFLAGS = -lstdc++

$(StaticLib) : $(Objs)
    ar rcs $@ $^

$(DynamicLib) : $(Objs)
    g++ -shared -o $@ $^ $(LDFLAGS)

$(Bin) : $(Objs)
    g++ $(Objs) -o $@

%.d : %.cpp
    $(CC) -MT"$(<:.cpp=.o) $@" -MM $(CXXFLAGS) $(CPPFLAGS) $< > $@

sinclude $(Deps)
```

```
.PHONY : clean
clean:
    rm -f $(Objs) $(Deps) $(StaticLib) $(DynamicLib) $(Bin)
```

1.2 makefile 操作系统检测方法

使用两个简单的技巧检测操作系统：

1. 首先是环境变量 OS
2. 然后 uname 命令

```
ifeq ($(OS),Windows_NT)     # is Windows_NT on XP, 2000, 7, Vista, 10...
    detected_OS := Windows
else
    detected_OS := $(shell uname)  # same as "uname -s"
endif
```

或者更安全的方式，如果不是在 Windows 上并且 uname 不可用：

```
ifeq ($(OS),Windows_NT)
    detected_OS := Windows
else
    detected_OS := $(shell sh -c 'uname 2>/dev/null || echo Unknown')
endif
```

如果你想区分 Cygwin / MinGW / MSYS / Windows，肯杰克逊提出了一个有趣的选择。看到他的答案看起来像这样：

```
ifeq '$(findstring ;,$(PATH))' ';'
    detected_OS := Windows
else
    detected_OS := $(shell uname 2>/dev/null || echo Unknown)
    detected_OS := $(patsubst CYGWIN%,Cygwin,$(detected_OS))
    detected_OS := $(patsubst MSYS%,MSYS,$(detected_OS))
    detected_OS := $(patsubst MINGW%,MSYS,$(detected_OS))
endif
```

然后您可以根据以下内容选择相关内容 detected_OS：

```
ifeq ($(detected_OS),Windows)
    CFLAGS += -D WIN32
endif
ifeq ($(detected_OS),Darwin)        # Mac OS X
    CFLAGS += -D OSX
endif
ifeq ($(detected_OS),Linux)
    CFLAGS   +=   -D LINUX
endif
ifeq ($(detected_OS),GNU)           # Debian GNU Hurd
    CFLAGS   +=   -D GNU_HURD
endif
ifeq ($(detected_OS),GNU/kFreeBSD)  # Debian kFreeBSD
    CFLAGS   +=   -D GNU_kFreeBSD
endif
ifeq ($(detected_OS),FreeBSD)
    CFLAGS   +=   -D FreeBSD
```

```
endif
ifeq ($(detected_OS),NetBSD)
    CFLAGS   +=   -D NetBSD
endif
ifeq ($(detected_OS),DragonFly)
    CFLAGS   +=   -D DragonFly
endif
ifeq ($(detected_OS),Haiku)
    CFLAGS   +=   -D Haiku
endif
```

笔记：

命令 uname 与 uname -s 因为 option -s（--kernel-name）是默认值相同。看看为什么 uname -s 比这更好 uname -o。

使用 OS（而不是 uname）简化了识别算法。您仍然可以单独使用 uname，但您必须处理 if/else 块以检查所有 MinGW，Cygwin 等变体。

环境变量 OS 始终设置为"Windows_NT" 不同的 Windows 版本（请参阅%OS%Wikipedia 上的环境变量）。

另一种方法 OS 是环境变量 MSVC（它检查 MS Visual Studio 的存在，请参阅使用 Visual C ++ 的示例）。

下面我提供一个使用 make 和 gcc 构建共享库的完整示例：*.so 或者 *.dll 取决于平台。这个例子尽可能简单易懂。

要在 Windows 上安装 make，gcc 请参阅 Cygwin 或 MinGW。

我的例子基于五个文件

```
lib
    Makefile
    hello.h
    hello.c
app
    Makefile
    main.c
```

提醒:Makefile 使用制表缩进。在示例文件下面复制粘贴时的注意事项。

这两个 Makefile 文件

1. lib/Makefile

```
ifeq ($(OS),Windows_NT)
    uname_S := Windows
else
    uname_S := $(shell uname -s)
endif

ifeq ($(uname_S), Windows)
    target = hello.dll
endif
ifeq ($(uname_S), Linux)
    target = libhello.so
endif
#ifeq ($(uname_S), .....) #See https://stackoverflow.com/a/27776822/938111
#    target = .....
#endif

%.o: %.c
    gcc  -c $<  -fPIC  -o $@
```

```
    # -c $<  => $< is first file after ':' => Compile hello.c
    # -fPIC  => Position-Independent Code (required for shared lib)
    # -o $@  => $@ is the target => Output file (-o) is hello.o

$(target): hello.o
    gcc  $^  -shared  -o $@
    # $^      => $^ expand to all prerequisites (after ':') => hello.o
    # -shared => Generate shared library
    # -o $@   => Output file (-o) is $@ (libhello.so or hello.dll)
```

2. app/Makefile

```
ifeq ($(OS),Windows_NT)
    uname_S := Windows
else
    uname_S := $(shell uname -s)
endif

ifeq ($(uname_S), Windows)
    target = app.exe
endif
ifeq ($(uname_S), Linux)
    target = app
endif
#ifeq ($(uname_S), .....) #See https://stackoverflow.com/a/27776822/938111
#    target = .....
#endif

%.o: %.c
    gcc  -c $< -I ../lib  -o $@
    # -c $<     => compile (-c) $< (first file after :) = main.c
    # -I ../lib => search headers (*.h) in directory ../lib
    # -o $@     => output file (-o) is $@ (target) = main.o

$(target): main.o
    gcc  $^  -L../lib  -lhello  -o $@
    # $^       => $^ (all files after the :) = main.o (here only one file)
    # -L../lib => look for libraries in directory ../lib
    # -lhello  => use shared library hello (libhello.so or hello.dll)
    # -o $@    => output file (-o) is $@ (target) = "app.exe" or "app"
```

要了解更多信息，请阅读 cfi 指出的自动变量文档。

源代码

- lib/hello.h

```
#ifndef HELLO_H_
#define HELLO_H_

const char* hello();

#endif
```

- lib/hello.c

```
#include "hello.h"
```

```
const char* hello()
{
    return "hello";
}
```

- app/main.c

```
#include "hello.h" //hello()
#include <stdio.h> //puts()

int main()
{
    const char* str = hello();
    puts(str);
}
```

构建

修复 Makefile (通过一个制表替换前导空格) 的复制粘贴。

```
> sed  's/^  */\t/'  -i  */Makefile
```

make 两个平台上的命令都是相同的。给定的输出是在类 Unix 操作系统上：

```
> make -C lib

  make: Entering directory '/tmp/lib'
  gcc  -c hello.c  -fPIC  -o hello.o
  # -c hello.c  => hello.c is first file after ':' => Compile hello.c
  # -fPIC       => Position-Independent Code (required for shared lib)
  # -o hello.o  => hello.o is the target => Output file (-o) is hello.o
  gcc  hello.o  -shared  -o libhello.so
  # hello.o        => hello.o is the first after ':' => Link hello.o
  # -shared        => Generate shared library
  # -o libhello.so => Output file (-o) is libhello.so (libhello.so or hello.dll)
  make: Leaving directory '/tmp/lib'

> make -C app
  make: Entering directory '/tmp/app'
  gcc  -c main.c -I ../lib  -o main.o
  # -c main.c => compile (-c) main.c (first file after :) = main.cpp
  # -I ../lib => search headers (*.h) in directory ../lib
  # -o main.o => output file (-o) is main.o (target) = main.o
  gcc  main.o  -L../lib  -lhello  -o app
  # main.o   => main.o (all files after the :) = main.o (here only one file)
  # -L../lib => look for libraries in directory ../lib
  # -lhello  => use shared library hello (libhello.so or hello.dll)
  # -o app   => output file (-o) is app.exe (target) = "app.exe" or "app"
  make: Leaving directory '/tmp/app'
```

运行

应用程序需要知道共享库的位置。

在 Windows 上，一个简单的解决方案是复制应用程序所在的库：

```
> cp -v lib/hello.dll app
`lib/hello.dll' -> `app/hello.dll'
```

在类 Unix 操作系统上，您可以使用 LD_LIBRARY_PATH 环境变量：

```
> export LD_LIBRARY_PATH=lib
```

在 Windows 上运行该命令：

```
> app/app.exe
hello
```

在类 Unix 操作系统上运行命令：

```
> app/app
hello
```

1.3 next

1.4 next