```
VERSION = 2
# 给变量 VERSION 赋值

PATCHLEVEL = 6
# 给变量 PATCHLEVEL 赋值

SUBLEVEL = 22
# 给变量 SUBLEVEL 赋值

EXTRAVERSION = .6
# 给变量 EXTRAVERSION 赋值

NAME = Holy Dancing Manatees, Batman!
# 给变量 NAME 赋值

# *DOCUMENTATION*
# To see a list of typical targets execute "make help"
# More info can be located in ./README
# Comments in this file are targeted only to the developer, do not
# expect to learn how to build the kernel reading this file.

# Do not:
# o use make's built-in rules and variables
#       (this increases performance and avoid hard-to-debug behavour);
# o print "Entering directory ...";
MAKEFLAGS += -rR --no-print-directory
# 操作符“+=”的作用是给变量（“+=”前面的 MAKEFLAGS）追加值。
# 如果变量（“+=”前面的 MAKEFLAGS）之前没有定义过，那么，“+=”会自动变成“=”；
# 如果前面有变量（“+=”前面的 MAKEFLAGS）定义，那么“+=”会继承于前次操作的赋值符；
# 如果前一次的是“:=”，那么“+=”会以“:=”作为其赋值符
# 在执行 make 时的命令行选项参数被通过变量 “MAKEFLAGS”传递给子目录下的 make 程序。
# 对于这个变量除非使用指示符“unexport”对它们进行声明，它们在整个 make 的执行过程中始终被自动的传递给所有的子 make。
# 还有个特殊变量 SHELL 与 MAKEFLAGS 一样，默认情况（没有用“unexport”声明）下在整个 make 的执行过程中被自动的传递给所有的子 make。
#
# -rR --no-print-directory
# -r disable the built-in impilict rules.
# -R disable the built-in variable setttings.
# --no-print-directory。
```

```
# We are using a recursive build, so we need to do a little thinking
# to get the ordering right.
#
# Most importantly: sub-Makefiles should only ever modify files in
# their own directory. If in some directory we have a dependency on
# a file in another dir (which doesn't happen often, but it's often
# unavoidable when linking the built-in.o targets which finally
# turn into vmlinux), we will call a sub make in that other dir, and
# after that we are sure that everything which is in that other dir
# is now up to date.
#
# The only cases where we need to modify files which have global
# effects are thus separated out and done before the recursive
# descending is started. They are now explicitly listed as the
# prepare rule.

# To put more focus on warnings, be less verbose as default
# Use 'make V=1' to see the full commands

ifdef V
ifeq ("$(origin V)", "command line")
KBUILD_VERBOSE = $(V)
endif
endif
ifndef KBUILD_VERBOSE
KBUILD_VERBOSE = 0
endif
# “ifdef”是条件关键字。语法是 ifdef <variable-name>; <text-if-true>;
else <text-if-false>; endif
# ifdef 只检验一个变量是否被赋值，它并不会去推导这个变量，并不会把变量
扩展到当前位置。
# “ifeq”与“ifdef”类似。
# “ifeq”语法是 ifeq (<arg1>;, <arg2>;)，功能是比较参数“arg1”和
“arg2”的值是否相同。
#
# 函数 origin 并不操作变量的值，只是告诉你你的这个变量是哪里来的。
# 语法是： $(origin <variable>;)
# origin 函数的返回值有：
# “undefined”从来没有定义过、“default”是一个默认的定义、
“environment”是一个环境变量、
# “file”这个变量被定义在 Makefile 中、“command line”这个变量是被命
令行定义的、
# “override”是被 override 指示符重新定义的、“automatic”是一个命令运
```

行中的自动化变量
#
应用变量的语法是：$(变量名)。如 KBUILD_VERBOSE = $(V)中的$(V)。
#
KBUILD_VERBOSE 的值根据在命令行中是否定义了变量 V，
当没有定义时，默认为 V＝0，输出为 short version；可以用 make V=1 来输出全部的命令。
#
ifndef 与 ifdef 语法类似，但功能恰好相反。ifndef 是判断变量是不是没有被赋值。

Call a source code checker (by default, "sparse") as part of the
C compilation.
#
Use 'make C=1' to enable checking of only re-compiled files.
Use 'make C=2' to enable checking of *all* source files, regardless
of whether they are re-compiled or not.
#
See the file "Documentation/sparse.txt" for more details, including
where to get the "sparse" utility.

ifdef C
ifeq ("$(origin C)", "command line")
KBUILD_CHECKSRC = $(C)
endif
endif
ifndef KBUILD_CHECKSRC
KBUILD_CHECKSRC = 0
endif
# ifdef 是 Makefile 的条件关键字，其语法是：ifdef <variable-name>;
# 如果变量<variable-name>;的值非空，那到表达式为真。否则，表达式为假。
# ifndef 也是 Makefile 的条将关键字，功能与 ifdef 相反，语法相似。

Use make M=dir to specify directory of external module to build
Old syntax make ... SUBDIRS=$PWD is still supported
Setting the environment variable KBUILD_EXTMOD take precedence
ifdef SUBDIRS
KBUILD_EXTMOD ?= $(SUBDIRS)
endif
ifdef M
ifeq ("$(origin M)", "command line")
KBUILD_EXTMOD := $(M)
endif
endif

```
# ifdef 是 Makefile 的条件关键字，其语法是：ifdef <variable-name>;
# 如果变量<variable-name>;的值非空，那到表达式为真。否则，表达式为假。
#
# ifeq 是 Makefile 的条件关键字，其语法是：ifeq (<arg1>;, <arg2>;)，比较参数“arg1”和“arg2”的值是否相同。
#
# 操作符“：=”与操作符“+=”的功能相同，只是操作符“：=”后面的用来定义变量（KBUILD_EXTMOD）的变量 M 只能是前面定义好的，
# 如果操作符“？=”前面的变量 KBUILD_EXTMOD 没有定义过，那么就将 SUBDIRS 赋给 KBUILD_EXTMOD;
# 如果定义过，则语句 KBUILD_EXTMOD ?= $(SUBDIRS)什么也不做。

# kbuild supports saving output files in a separate directory.
# To locate output files in a separate directory two syntaxes are supported.
# In both cases the working directory must be the root of the kernel src.
# 1) O=
# Use "make O=dir/to/store/output/files/"
#
# 2) Set KBUILD_OUTPUT
# Set the environment variable KBUILD_OUTPUT to point to the directory
# where the output files shall be placed.
# export KBUILD_OUTPUT=dir/to/store/output/files/
# make
#
# The O= assignment takes precedence over the KBUILD_OUTPUT environment
# variable.


# KBUILD_SRC is set on invocation of make in OBJ directory
# KBUILD_SRC is not intended to be used by the regular user (for now)
ifeq ($(KBUILD_SRC),)
# ifeq 是 Makefile 的条件关键字，其语法是：ifeq (<arg1>;, <arg2>;)，比较参数“arg1”和“arg2”的值是否相同。

# OK, Make called in directory where kernel src resides
# Do we want to locate output files in a separate directory?
ifdef O
ifeq ("$(origin O)", "command line")
KBUILD_OUTPUT := $(O)
endif
endif
# ifdef 是 Makefile 的条件关键字，其语法是：ifdef <variable-name>;
# 如果变量<variable-name>;的值非空，那到表达式为真。否则，表达式为假。
# ifeq 是 Makefile 的条件关键字，其语法是：ifeq (<arg1>;, <arg2>;)，比较参数“arg1”和“arg2”的值是否相同。
```

```
# origin 是 Makefile 的一个判别变量是哪里来的函数，其语法是：$(origin <variable>;)


# That's our default target when none is given on the command line
PHONY := _all
_all:
# 为变量 PHONY 追加_all
# Makefile 的规则：
# 目标：依赖文件
# 命令 1
# 命令 2
# ...
#
# 没有依赖文件的目标称为“伪目标”。伪目标并不是一个文件，只是一个标签。
# 由于伪目标不是一个文件，所以 make 无法生成它的依赖关系和决定它是否要执行，
# 只有在命令行中输入（即显示地指明）这个“目标”才能让其生效，此处为"make _all"。

ifneq ($(KBUILD_OUTPUT),)
# ifneq 是 Makefile 的条件关键字，其语法是：ifneq (<arg1>;, <arg2>;)，
# 功能是：比较参数“arg1”和“arg2”的值是否不相同，功能与 ifeq 相反。
# Invoke a second make in the output directory, passing relevant variables
# check that the output directory actually exists
saved-output := $(KBUILD_OUTPUT)
KBUILD_OUTPUT := $(shell cd $(KBUILD_OUTPUT) && /bin/pwd)
# 函数 shell 是 make 与外部环境的通讯工具，它用于命令的扩展。
# shell 函数起着调用 shell 命令（cd $(KBUILD_OUTPUT) && /bin/pwd）和返回命令输出结果的参数的作用。
# Make 仅仅处理返回结果，再返回结果替换调用点之前，make 将每一个换行符或者一对回车/换行符处理为单个空格；
# 如果返回结果最后是换行符（和回车符），make 将把它们去掉。

$(if $(KBUILD_OUTPUT),, \
$(error output directory "$(saved-output)" does not exist))
# 函数 if 对在函数上下文中扩展条件提供了支持（相对于 GNU make makefile 文件中的条件语句，例如 ifeq 指令。）
# if 函数的语法是：$(if <condition>,<then-part>) 或是 $(if <condition>,<then-part>,<else-part>)。
# 如果条件$(KBUILD_OUTPUT)为真（非空字符串），那么两个逗号之间的空字符（注意连续两个逗号的作用）将会是整个函数的返回值，
# 如果$(KBUILD_OUTPUT)为假（空字符串），那么$(error output directory "$(saved-output)" does not exist）会是整个函数的返回值，
# 此时如果<else-part>没有被定义，那么，整个函数返回空字串。
```

```
#
# 函数 error 的语法是：$(error <text ...>;)
# 函数 error 的功能是：产生一个致命的错误，output directory
"$(saved-output)" does not exist 是错误信息。
# 注意，error 函数不会在一被使用就会产生错误信息，所以如果你把其定义在某个变量中，并在后续的脚本中使用这个变量，那么也是可以的。
#
# 命令“$(if $(KBUILD_OUTPUT),, \”中最后的“\”的作用是：紧接在“\”下面的“哪一行”的命令是“\”所在行的命令的延续。
# 如果要让前一个命令的参数等应用与下一个命令，那么这两个命令应该写在同一行，如果一行写不下两个命令，可以在第一行末尾添上符号"\"，然后在下一行接着写。
# 如果是几个命令写在同一行，那么后面的命令是在前面命令的基础上执行。如 cd /     ls 这两个命令写在同一行，那么 ls 显示的是根目录/下的文件和文件夹。

PHONY += $(MAKECMDGOALS)
# 将变量 KBUILD_OUTPUT 的值追加给变量 saved-output，

$(filter-out _all,$(MAKECMDGOALS)) _all:
$(if $(KBUILD_VERBOSE:1=),@)$(MAKE) -C $(KBUILD_OUTPUT) \
KBUILD_SRC=$(CURDIR) \
KBUILD_EXTMOD="$(KBUILD_EXTMOD)" -f $(CURDIR)/Makefile $@
# 反过滤函数——filter-out，语法是：$(filter-out <pattern...>;,<text>;)
# 函数 filter-out 的功能是：去掉$(MAKECMDGOALS）中符合规则_all 的所有字符串后，剩下的作为返回值。
# 函数 filter-out 调用与伪目标_all 在同一行。
# 伪目标_all 下面的以 tab 开头的三行是命令，因为每行最后都有"\"，所以这三行命令应该是写在同一行的，即后面的命令要受到处于它之前的那些命令的影响。
#
# $(if $(KBUILD_VERBOSE:1=),@) 含义是如果$(KBUILD_VERBOSE:1=) 不为空，则等于$@
# 自动化变量"$@"表示规则中的目标文件集，在模式规则中，如果有多个目标，那么，"$@"就是匹配于目标中模式定义的集合。
# 自动化变量还有"$<"、"$%"、"$<"等。
#
# 宏变量$（MAKE）的值为 make 命令和参数（参数可省）。
#
# 执行命令 KBUILD_SRC=$(CURDIR)的结果是把变量 CURDIR 的值赋给变量 KBUILD_SRC。
# CURDIR 这个变量是 Makefile 提供的，代表了 make 当前的工作路径。

# Leave processing to above invocation of make
skip-makefile := 1
```

```
endif # ifneq ($(KBUILD_OUTPUT),)
endif # ifeq ($(KBUILD_SRC),)
# 给变量 skip-makefile 追加值 1.
# 命令 endif # ifneq ($(KBUILD_OUTPUT),)的意思是这一行的 endif 与 ifneq
($(KBUILD_OUTPUT),)相对应，
# 其实它本身已经解释清楚了，我只是让他变得明显一点而已。
# 命令 endif # ifeq ($(KBUILD_SRC),)的意思是这一行的 endif 与 ifeq
($(KBUILD_SRC),)相对应。

# We process the rest of the Makefile if this is the final invocation of
make
ifeq ($(skip-makefile),)
# 判断变量 skip-makefile 与空字符是否相同,即判断变量 skip-makefile 的值
是否为空。

# If building an external module we do not care about the all: rule
# but instead _all depend on modules
PHONY += all
ifeq ($(KBUILD_EXTMOD),)
_all: all
else
_all: modules
endif
# 为变量 PHONY 追加值 all。
# 判断变量 KBUILD_EXTMOD 的值与空字符是否相同，即判断变量 KBUILD_EXTMOD
的值是否为空。
# 定义两种不同情况下使用的规则_all: all 和_all: modules

srctree     := $(if $(KBUILD_SRC),$(KBUILD_SRC),$(CURDIR))
TOPDIR     := $(srctree)
# FIXME - TOPDIR is obsolete, use srctree/objtree
# 调用 if 函数，根据变量 KBUILD_SRC 的值是否为空，决定将变量 KBUILD_SRC
或者变量 CURDIR 的值赋给变量 srctree
# 为变量 TOPDIR 追加变量 srctree 的值
objtree     := $(CURDIR)
src     := $(srctree)
obj     := $(objtree)

VPATH     := $(srctree)$(if $(KBUILD_EXTMOD),:$(KBUILD_EXTMOD))
# “VPATH”是 Makefile 文件中的特殊变量。
# ，如果没有指明这个变量，make 只会在当前的目录中去找寻依赖文件和目标
文件。
# 如果定义了这个变量，那么，make 就会在当当前目录找不到的情况下，到所
指定的目录中去找寻文件了。
```

```
export srctree objtree VPATH TOPDIR
# 为变量 objtree、src、obj 分别追加变量 CURDIR、srctree、objtree 的值
# make 使用“VPATH”变量来指定“依赖文件”的搜索路径。
# 为变量 VPATH 追加变量 VPATH 的值
# 关键词 export 用来声明变量，被声明的变量要被传递到下级 Makefile 中。
# export srctree objtree VPATH TOPDIR 声明了四个变量，这四个变量在 make
嵌套时都将被传递到下级 Makefile。


# SUBARCH tells the usermode build what the underlying arch is. That is
set
# first, and if a usermode build is happening, the "ARCH=um" on the command
# line overrides the setting of ARCH below. If a native build is happening,
# then ARCH is assigned, getting whatever value it gets normally, and
# SUBARCH is subsequently ignored.

SUBARCH := $(shell uname -m | sed -e s/i.86/i386/ -e s/sun4u/sparc64/ \
-e s/arm.*/arm/ -e s/sa110/arm/ \
-e s/s390x/s390/ -e s/parisc64/parisc/ \
-e s/ppc.*/powerpc/ -e s/mips.*/mips/ )
# 为变量 SUBARCH 追加调用 shell 执行 sed 后的返回值。
# sed 是一种在线编辑器，它一次处理一行内容。
# Sed 主要用来自动编辑一个或多个文件；简化对文件的反复操作；编写转换程
序等。


# Cross compiling and selecting different set of gcc/bin-utils
#
---------------------------------------------------------------------
------
#
# When performing cross compilation for other architectures ARCH shall
be set
# to the target architecture. (See arch/* for the possibilities).
# ARCH can be set during invocation of make:
# make ARCH=ia64
# Another way is to have ARCH set in the environment.
# The default ARCH is the host where make is executed.

# Cross compiling and selecting different set of gcc/bin-utils
#
---------------------------------------------------------------------
------
#
# When performing cross compilation for other architectures ARCH shall
```

```
be set
# to the target architecture. (See arch/* for the possibilities).
# ARCH can be set during invocation of make:
# make ARCH=ia64
# Another way is to have ARCH set in the environment.
# The default ARCH is the host where make is executed.

# CROSS_COMPILE specify the prefix used for all executables used
# during compilation. Only gcc and related bin-utils executables
# are prefixed with $(CROSS_COMPILE).
# CROSS_COMPILE can be set on the command line
# make CROSS_COMPILE=ia64-linux-
# Alternatively CROSS_COMPILE can be set in the environment.
# Default value for CROSS_COMPILE is not to prefix executables
# Note: Some architectures assign CROSS_COMPILE in their arch/*/Makefile
# 上面已经说了，下面的这些是用于交叉编译（嵌入式 linux 的编译环境就是交叉编译）。
# 如果你不清楚嵌入式 linux 是什么，但是你又想知道它，记住：www.baidu.com

ARCH      ?= $(SUBARCH)
CROSS_COMPILE ?=
# 变量 ARCH 用来指明目标 cpu 的构架
# 设置变量 ARCH 的方法有两种，
# 一是：在命令行中 如：make ARCH=ia64；
# 二是：设置环境变量，在环境变量中默认的 ARCH 的值是执行 make 的 cpu 构架
# 不论怎么弄，目的就是使编译出来的目标文件（可执行文件）面向的是你的目标平台（在嵌入式开发中）。
#
# 操作符“?=”的作用是：如果 ARCH 未被定义过，那么将变量 SUBARCH 的值赋给变量 ARCH，
# 如果变量 ARCH 已经被定义过，那么这条语句什么也不做。
#
# ARCH 指定在嵌入式开发中你的目标板上的 cpu 类型（构架），如：arm，ppc，powerpc 等
# 变量 CROSS_COMPILE 指定交叉编译用的交叉编译器，这里的 CROSS_COMPILE 就是让你指定交叉编译器的路径。
# 如果你设置好了 PATH 那么直接把这句加上就可以,如果没有那么请指定路径，

# Architecture as present in compile.h
UTS_MACHINE := $(ARCH)
# 将变量 ARCH 直接展开给变量 UTS_MACHINE。

KCONFIG_CONFIG ?= .config
# 在变量 KCONFIG_CONFIG 没赋值的情况下，将.config 赋给变量 KCONFIG_CONFIG；如果已经赋值，那么什么也不做。
```

```
# SHELL used by kbuild
CONFIG_SHELL := $(shell if [ -x "$$BASH" ]; then echo $$BASH; \
else if [ -x /bin/bash ]; then echo /bin/bash; \
else echo sh; fi ; fi)
# 将生成 shell 程序来执行 if 函数后返回的结果展开给变量 CONFIG_SHELL;
# if 函数中，在 else 中又嵌套了 if 函数。

HOSTCC       = gcc
HOSTCXX      = g++
HOSTCFLAGS     = -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer
HOSTCXXFLAGS = -O2
# 分别为变量 HOSTCC 、HOSTCXX 、HOSTCFLAGS 、HOSTCXXFLAGS 赋值。

# Decide whether to build built-in, modular, or both.
# Normally, just do built-in.

KBUILD_MODULES :=
KBUILD_BUILTIN := 1
# 分别为变量 KBUILD_MODULES、KBUILD_BUILTIN 赋值。

# If we have only "make modules", don't compile built-in objects.
# When we're building modules with modversions, we need to consider
# the built-in objects during the descend as well, in order to
# make sure the checksums are up to date before we record them.

ifeq ($(MAKECMDGOALS),modules)
KBUILD_BUILTIN := $(if $(CONFIG_MODVERSIONS),1)
endif
# ifeq 判断变量 MAKECMDGOALS 的值与 modules 是否相同;
# 第二行将调用 if 后返回的值展开给变量 KBUILD_BUILTIN;
# if 函数判断变量 CONFIG_MODVERSIONS 的值是否为非空字符串，如果是非空字
符串，则执行逗号后面的 1
# If we have "make <whatever> modules", compile modules
# in addition to whatever we do anyway.
# Just "make" or "make all" shall build modules as well

ifneq ($(filter all _all modules,$(MAKECMDGOALS)),)
KBUILD_MODULES := 1
endif
# ifneq 判断调用函数 filter 的返回值与空字符串是否相同。
# 如果相同，那么执行第二行，即把 1 赋给变量 KBUILD_MODULES

ifeq ($(MAKECMDGOALS),)
KBUILD_MODULES := 1
endif
```

```
# ifeq 判断变量 MAKECMDGOALS 的值是否与空字符相同。如果相同，则执行第二行；
# 第二行是把 1 赋给变量 KBUILD_MODULES。
export KBUILD_MODULES KBUILD_BUILTIN
export KBUILD_CHECKSRC KBUILD_SRC KBUILD_EXTMOD
# 用关键词 export 声明变量 KBUILD_MODULES KBUILD_BUILTIN、KBUILD_CHECKSRC KBUILD_SRC KBUILD_EXTMOD。
# 如果用关键词 export 声明了变量，那么被声明的变量将会被传递到下级 Makefile 中。

# Beautify output
#
---------------------------------------------------------------------
------
#
# Normally, we echo the whole command before executing it. By making
# that echo $($(quiet)$(cmd)), we now have the possibility to set
# $(quiet) to choose other forms of output instead, e.g.
#
#                 quiet_cmd_cc_o_c = Compiling $(RELDIR)/$@
#                 cmd_cc_o_c             = $(CC) $(c_flags) -c -o $@ $<
#
# If $(quiet) is empty, the whole command will be printed.
# If it is set to "quiet_", only the short version will be printed.
# If it is set to "silent_", nothing will be printed at all, since
# the variable $(silent_cmd_cc_o_c) doesn't exist.
#
# A simple variant is to prefix commands with $(Q) - that's useful
# for commands that shall be hidden in non-verbose mode.
#
# $(Q)ln $@ :<
#
# If KBUILD_VERBOSE equals 0 then the above command will be hidden.
# If KBUILD_VERBOSE equals 1 then the above command is displayed.

ifeq ($(KBUILD_VERBOSE),1)
quiet =
Q =
else
quiet=quiet_
Q = @
endif
# 函数 ifeq 判断变量 KBUILD_VERBOSE 的值与 1 是否相同。如果相同则执行第二
```

三行，否则执行五六行。
二三五六行都是为变量赋值。

```
# If the user is running make -s (silent mode), suppress echoing of
# commands

ifneq ($(findstring s,$(MAKEFLAGS)),)
quiet=silent_
endif
```

函数 ifneq 判断函数 findstring 返回值与空字符是否相同，如果相同则执行第二行（将 silent_赋给变量 quiet）。
函数 findstring 的语法是：$(findstring <find>;,<in>;)
函数 findstring 的功能是：如果在$(MAKEFLAGS)中能找到字符 s，那么返回字符 s；否则返回空字符。

```
export quiet Q KBUILD_VERBOSE
```

用关键词 export 声明变量 quiet Q KBUILD_VERBOSE，使得它们能被传到下级 Makefile。

```
# Look for make include files relative to root of kernel src
MAKEFLAGS += --include-dir=$(srctree)
```

给变量 MAKEFLAGS 追加--include-dir=$(srctree)。
--include-dir 是 make 的参数，用来指定一个被包含 makefile 的搜索目标。
也可以使用多个“-I”参数来指定多个目录。

```
# We need some generic definitions.
include $(srctree)/scripts/Kbuild.include
```

在 Makefile 使用 include 关键字可以把别的 Makefile 包含进来。
这很像 C 语言的#include，被包含的文件会原模原样的放在当前文件的包含位置。
include 的语法是： include <filename>;

```
# Make variables (CC, etc...)

AS      = $(CROSS_COMPILE)as
LD      = $(CROSS_COMPILE)ld
CC      = $(CROSS_COMPILE)gcc
CPP     = $(CC) -E
AR      = $(CROSS_COMPILE)ar
NM      = $(CROSS_COMPILE)nm
STRIP       = $(CROSS_COMPILE)strip
OBJCOPY     = $(CROSS_COMPILE)objcopy
OBJDUMP     = $(CROSS_COMPILE)objdump
AWK     = awk
GENKSYMS = scripts/genksyms/genksyms
```

```
DEPMOD     = /sbin/depmod
KALLSYMS = scripts/kallsyms
PERL     = perl
CHECK     = sparse

CHECKFLAGS         := -D__linux__ -Dlinux -D__STDC__ -Dunix -D__unix__
-Wbitwise $(CF)
MODFLAGS = -DMODULE
CFLAGS_MODULE     = $(MODFLAGS)
AFLAGS_MODULE     = $(MODFLAGS)
LDFLAGS_MODULE = -r
CFLAGS_KERNEL =
AFLAGS_KERNEL =
# 给一系列变量赋值。


# Use LINUXINCLUDE when you must reference the include/ directory.
# Needed to be compatible with the O= option
LINUXINCLUDE       := -Iinclude \
$(if $(KBUILD_SRC),-Iinclude2 -I$(srctree)/include) \
-include include/linux/autoconf.h
# 为变量 LINUXINCLUDE 赋值。

CPPFLAGS               := -D__KERNEL__ $(LINUXINCLUDE)

CFLAGS                   := -Wall -Wundef -Wstrict-prototypes
-Wno-trigraphs \
-fno-strict-aliasing -fno-common
AFLAGS                   := -D__ASSEMBLY__
# 给 FLAGS 系列变量赋值。

# Read KERNELRELEASE from include/config/kernel.release (if it exists)
KERNELRELEASE = $(shell cat include/config/kernel.release 2> /dev/null)
KERNELRELEASE = $(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)
# 给变量 KERNELRELEASE、KERNELRELEASE 赋值。
# >是重定向符号。
# 第一行中 cat 和>连用的作用是：将文件 include/config/kernel.release 的
内容写入到文件/dev/null 中。

export VERSION PATCHLEVEL SUBLEVEL KERNELRELEASE KERNELVERSION
export ARCH CONFIG_SHELL HOSTCC HOSTCFLAGS CROSS_COMPILE AS LD CC
export CPP AR NM STRIP OBJCOPY OBJDUMP MAKE AWK GENKSYMS PERL UTS_MACHINE
export HOSTCXX HOSTCXXFLAGS LDFLAGS_MODULE CHECK CHECKFLAGS
```

```
export CPPFLAGS NOSTDINC_FLAGS LINUXINCLUDE OBJCOPYFLAGS LDFLAGS
export CFLAGS CFLAGS_KERNEL CFLAGS_MODULE
export AFLAGS AFLAGS_KERNEL AFLAGS_MODULE
# 用关键词 export 声明变量，使得这些变量能传到下级 Makefile 中

# When compiling out-of-tree modules, put MODVERDIR in the module
# tree rather than in the kernel tree. The kernel tree might
# even be read-only.
export MODVERDIR := $(if $(KBUILD_EXTMOD),$(firstword
$(KBUILD_EXTMOD))/).tmp_versions
# 用关键词 export 声明变量 MODVERDIR，使得变量 MODVERDIR 能传到下级
Makefile 中
# 将 if 函数的返回值展开后与.tmp_versions 共同赋给变量 MODVERDIR。
# if 函数判断变量 KBUILD_EXTMOD 的值是否为空字符串，如果不为空字符串，
执行函数 firstword。
# 函数 firstword 的语法是：$(firstword <text>;)。
# 函数 firstword 在此处的功能是：取出变量 KBUILD_EXTMOD 的第一个字符。

# Files to ignore in find ... statements

RCS_FIND_IGNORE := \( -name SCCS -o -name BitKeeper -o -name .svn -o -name
CVS -o -name .pc -o -name .hg -o -name .git \) -prune -o
export RCS_TAR_IGNORE := --exclude SCCS --exclude BitKeeper
--exclude .svn --exclude CVS --exclude .pc --exclude .hg --exclude .git
# 分别为变量 RCS_FIND_IGNORE、RCS_FIND_IGNORE 赋值。
# 第二行中用关键词 export 声明变量 RCS_TAR_IGNORE，使它能够传递到下级
Makefile 中。
# 反斜杠\在第一行中只有转义的作用，不表示续行。
# 即\后面的“（”和“）”都不表示前括号和后括号，而是两个字符。
#-name 是命令 find 的一个参数，参数-name 用来指明要搜索的文件的部分或者
全名。
#-o 是 OR 运算符，语法是：表达式-o 表达式、如果第一个表达式是真，就不
会对第二个表达式求值。

#
==========================================================================
======
# Rules shared between *config targets and build targets

# Basic helpers built in scripts/
PHONY += scripts_basic
scripts_basic:
$(Q)$(MAKE) $(build)=scripts/basic
# 为变量 PHONY 追加 scripts_basic。
# 定义了一个伪目标 scripts_basic，第三行是针对伪目标 scripts_basic 要执
```

行的命令。
第三行是将变量展开，展开后是一个命令。

```
# To avoid any implicit rule to kick in, define an empty command.
scripts/basic/%: scripts_basic ;
# 定义了一个规则，依赖关系为 scripts/basic/%: scripts_basic
# 冒号：的作用是定义一个空命令（正如上面解释所说）。

PHONY += outputmakefile
# 为变量 PHONY 追加 outputmakefile。
# outputmakefile generates a Makefile in the output directory, if using
a
# separate output directory. This allows convenient use of make in the
# output directory.
outputmakefile:
ifneq ($(KBUILD_SRC),)
$(Q)$(CONFIG_SHELL) $(srctree)/scripts/mkmakefile \
$(srctree) $(objtree) $(VERSION) $(PATCHLEVEL)
endif
# 定义一个伪目标 outputmakefile。
# 将三四行中所有的变量展开并且合并成一行命令。

# To make sure we do not include .config for any of the *config targets
# catch them early, and hand them over to scripts/kconfig/Makefile
# It is allowed to specify more targets when calling make, including
# mixing *config targets and build targets.
# For example 'make oldconfig all'.
# Detect when mixed targets is specified, and make a second invocation
# of make so .config is not included in this case either (for *config).

no-dot-config-targets := clean mrproper distclean \
cscope TAGS tags help %docs check% \
include/linux/version.h headers_% \
kernelrelease kernelversion
# 定义了一个依赖关系，但是没有命令。

config-targets := 0
mixed-targets := 0
dot-config         := 1
# 给变量 config-targets、mixed-targets、dot-config 赋值。


ifneq ($(filter $(no-dot-config-targets), $(MAKECMDGOALS)),)
ifeq ($(filter-out $(no-dot-config-targets), $(MAKECMDGOALS)),)
dot-config := 0
```

```
endif
endif
# 条件关键词 ifneq 判断 filter 函数的返回值与变量 MAKECMDGOALS 是否相同，如果相同，则执行第二行。
# 第二行，条件关键词 ifeq 判断函数 filter-out 返回值与变量 MAKECMDGOALS 是否相同，如果相同则执行第三行。
# 函数 filter 的语法是：$(filter <pattern...>;,<text>;)。
# 功能是：返回 $(MAKECMDGOALS)中符合模式$(no-dot-config-targets)的字符串
# 函数 filter-out 与函数 filter 在语法是相似，在功能上恰好相反。
# 函数 filter-out 返回字符串$(MAKECMDGOALS)中不符合模式
$(no-dot-config-targets)的字符串。
# 第三行是给变量 dot-config 赋值 0.


ifeq ($(KBUILD_EXTMOD),)
ifneq ($(filter config %config,$(MAKECMDGOALS)),)
config-targets := 1
ifneq ($(filter-out config %config,$(MAKECMDGOALS)),)
mixed-targets := 1
endif
endif
endif
# 条件关键词 ifeq 判断 KBUILD_EXTMOD 的值与空格是否相同，如果是则执行第二行。
# 条件关键词 ifneq 判断函数 filter 的返回值与空格是否不相同，如果不相同，则执行第三行。
# 函数 filter 返回字符串$(MAKECMDGOALS))中符合模式 config %config 的字符串。
# 第三行为变量 config-targets 赋值 1
# 条件关键词 ifneq 判断函数 filter-out 返回值与空格是否不相同，如果不过不相同则执行第五行。
# 函数 filter-out 返回字符串$(MAKECMDGOALS)中不符合模式 config %config 的字符串。
# 第五行为变量 mixed-targets 赋值 1.


ifeq ($(mixed-targets),1)
# 条件关键词 ifeq 判断变量 mixed-targets 的值与 1 是否相同，即是否为 1
#
================================================================
======
# We're called with mixed targets (*config and build targets).
# Handle them one by one.
```

```
%:: FORCE
$(Q)$(MAKE) -C $(srctree) KBUILD_SRC= $@
# 定义了目标“%:”，依赖文件 FORCE
# 第二行，将所有变量展开后组成一行命令。

ifeq ($(mixed-targets),1)
# 条件关键词 ifeq 判断变量 mixed-targets 的值与 1 是否相同，即是否为 1
#
================================================================================
# We're called with mixed targets (*config and build targets).
# Handle them one by one.
else
ifeq ($(config-targets),1)
# 如果变量 mixed-targets 的值与 1 不相同，则执行 else 后面的（即执行第二行）。
# 条件关键词 ifeq 判断变量 config-targets 的值与 1 是否相同。
#
================================================================================
# *config targets only - make sure prerequisites are updated, and descend
# in scripts/kconfig to make the *config target

# Read arch specific Makefile to set KBUILD_DEFCONFIG as needed.
# KBUILD_DEFCONFIG may point out an alternative default configuration
# used for 'make defconfig'
include $(srctree)/arch/$(ARCH)/Makefile
export KBUILD_DEFCONFIG
# 如果变量 config-targets 的值与 1 相同，则执行从第一行开始直到遇到下面的 else 结束。
# 关键词 include 将文件$(srctree)/arch/$(ARCH)/Makefile 包含进来，
# 文件$(srctree)/arch/$(ARCH)/Makefile 被原模原样的放在当前文件的包含位置。
# 关键词 export 声明变量 KBUILD_DEFCONFIG 使得变量 KBUILD_DEFCONFIG 能传到下级 Makefile 中。

config %config: scripts_basic outputmakefile FORCE
$(Q)mkdir -p include/linux include/config
$(Q)$(MAKE) $(build)=scripts/kconfig $@
# 定义了一个规则，目标位 config %config 其中有个通配符%。
# 通配符%只能代表一个字符，config %config 可以表示 xconfig 和 gconfig、menuconfig 和 oldconfig 等。
# 二三行将变量展开后就规则得到要执行的命令。
```

```
else
# 如果变量 config-targets 的值与 1 不相同，则执行下面的
#
================================================================
======
# Build targets only - this includes vmlinux, arch specific targets, clean
# targets and others. In general all targets except *config targets.

ifeq ($(KBUILD_EXTMOD),)
# 条件关键词 ifeq 判断变量 KBUILD_EXTMOD 的值与空是否相同。
# Additional helpers built in scripts/
# Carefully list dependencies so we do not try to build scripts twice
# in parallel
PHONY += scripts
scripts: scripts_basic include/config/auto.conf
$(Q)$(MAKE) $(build)=$(@)
# 第一行给变量 PHONY 追加 scripts。
# 定义了一个规则。目标是 scripts，依赖文件是 scripts_basic
include/config/auto.conf
# 将第三行的所有命令展开就得到规则的命令。

# Objects we will link into vmlinux / subdirs we need to visit
init-y     := init/
drivers-y := drivers/ sound/
net-y     := net/
libs-y     := lib/
core-y     := usr/
endif # KBUILD_EXTMOD
# 为变量赋值。
# 上面的 endif 与 ifeq ($(KBUILD_EXTMOD),)相对应。

ifeq ($(dot-config),1)
# Read in config
-include include/config/auto.conf
# 条件关键词 ifeq 判断变量 dot-config 的值与 1 是否相同，如果相同这执行上
面这一条。
# 关键词 include 将文件 include/config/auto.conf 包含进来，该文件会原模
原样的放在当前文件的包含位置。
# 关键词 include 前面的减号的作用是让 make 不理那些无法读取的文件，而继
续执行。

ifeq ($(KBUILD_EXTMOD),)
# Read in dependencies to all Kconfig* files, make sure to run
# oldconfig if changes are detected.
-include include/config/auto.conf.cmd
```

条件关键词 ifeq 判断变量 KBUILD_EXTMOD 的值与 0 是否相同，如果相同这执行上面这一条。
关键词 include 将文件 include/config/auto.conf.cmd 包含进来，该文件会原模原样的放在当前文件的包含位置。
关键词 include 前面的减号的作用是让 make 不理那些无法读取的文件，而继续执行。

```
# To avoid any implicit rule to kick in, define an empty command
$(KCONFIG_CONFIG) include/config/auto.conf.cmd: ;
# 定义了一个规则，目标是$(KCONFIG_CONFIG)
include/config/auto.conf.cmd，没有依赖文件，命令。

# If .config is newer than include/config/auto.conf, someone tinkered
# with it and forgot to run make oldconfig.
# if auto.conf.cmd is missing then we are probably in a cleaned tree so
# we execute the config step to be sure to catch updated Kconfig files
include/config/auto.conf: $(KCONFIG_CONFIG)
include/config/auto.conf.cmd
$(Q)$(MAKE) -f $(srctree)/Makefile silentoldconfig
else
# 第一二行定义了一个规则。
# 目标是 include/config/auto.conf，依赖文件是$(KCONFIG_CONFIG)
include/config/auto.conf.cmd
# 将第二行的变量展开就得到命令。
# 上面的 else 与 ifeq ($(KBUILD_EXTMOD),)对应
# external modules needs include/linux/autoconf.h and
include/config/auto.conf
# but do not care if they are up-to-date. Use auto.conf to trigger the
test
PHONY += include/config/auto.conf
# 为变量 PHONY 追加值 include/config/auto.conf

include/config/auto.conf:
# 定义了一个伪目标
$(Q)test -e include/linux/autoconf.h -a -e $@ || (     \
echo;                   \
echo " ERROR: Kernel configuration is invalid.";     \
echo "                  include/linux/autoconf.h or $@ are missing."; \
echo "                  Run 'make oldconfig && make prepare' on kernel src
to fix it."; \
echo;                   \
/bin/false)
# 自动变量$@表示当前规则的目标变量名。
# echo 是 shell 中的显示命令，显示 echo 后面的字符串。
```

```
endif # KBUILD_EXTMOD
# 上面的 endif 与 ifeq ($(KBUILD_EXTMOD),)对应。
else
# Dummy target needed, because used as prerequisite
include/config/auto.conf: ;
endif # $(dot-config)
# 定义了一个伪目标 include/config/auto.conf，该规则没有依赖文件和命令。
# 上面的 endif 与 ifeq ($(dot-config),1)对应。

# The all: target is the default when no target is given on the
# command line.
# This allow a user to issue only 'make' to build a kernel including modules
# Defaults vmlinux but it is usually overridden in the arch makefile
all: vmlinux
# 定义了一个依赖关系，目标是 all，依赖文件是 vmlinux。

ifdef CONFIG_CC_OPTIMIZE_FOR_SIZE
CFLAGS     += -Os
else
CFLAGS     += -O2
endif
# 条件关键字 ifdef 只检验变量 CONFIG_CC_OPTIMIZE_FOR_SIZE 是否被赋值（非空）。
# 分别在两种情况下为变量 CFLAGS 追加不同的值。

include $(srctree)/arch/$(ARCH)/Makefile
# 用关键字声明文件$(srctree)/arch/$(ARCH)/Makefile，使得该文件能传递到下级 Makefile

ifdef CONFIG_FRAME_POINTER
CFLAGS     += -fno-omit-frame-pointer $(call
cc-option,-fno-optimize-sibling-calls,)
else
CFLAGS     += -fomit-frame-pointer
endif
# 条件关键字 ifdef 只检验变量 CONFIG_FRAME_POINTER 是否被赋值（非空）。
# 分别在两种情况下为变量 CFLAGS 追加不同的值。

ifdef CONFIG_DEBUG_INFO
CFLAGS     += -g
endif
# 条件关键字 ifdef 只检验变量 CONFIG_DEBUG_INFO 是否被赋值（非空）。
# 为变量 CFLAGS 追加-g
```

```
# Force gcc to behave correct even for buggy distributions
CFLAGS                   += $(call cc-option, -fno-stack-protector)
# 为变量 CFLAGS 追加$(call cc-option, -fno-stack-protector)。

# arch Makefile may override CC so keep this after arch Makefile is
included
NOSTDINC_FLAGS += -nostdinc -isystem $(shell $(CC)
-print-file-name=include)
CHECKFLAGS         += $(NOSTDINC_FLAGS)
# 分别为变量追加值
# 函数 shell 新生成一个 Shell 程序来执行由变量 CC 展开后形成的命令。

# warn about C99 declaration after statement
CFLAGS += $(call cc-option,-Wdeclaration-after-statement,)
# 为变量 CFLAGS 追加 call 函数返回的函数 call 的返回值。
# 函数 call 的语法:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)。
# 函数 call 的功能是：参数 cc-option 中的变量被字符串
-Wdeclaration-after-statement 和逗号后面的空格依次取代。
# 函数 call 的返回值是：被取代后的参数。

# disable pointer signed / unsigned warnings in gcc 4.0
CFLAGS += $(call cc-option,-Wno-pointer-sign,)
# 为变量 CFLAGS 追加 call 函数返回的函数 call 的返回值。
# 函数 call 的语法:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)。
## 函数 call 的功能是：参数 cc-option 中的变量被字符串-Wno-pointer-sign
和逗号后面的空格依次取代。

# Default kernel image to build when no specific target is given.
# KBUILD_IMAGE may be overruled on the command line or
# set in the environment
# Also any assignments in arch/$(ARCH)/Makefile take precedence over
# this default value
export KBUILD_IMAGE ?= vmlinux
# 用关键字声明了变量 KBUILD_IMAGE，使得该变量能传递到下级 Makefile 中。
# 符号“?=”在变量 KBUILD_IMAGE 没有赋值的情况下给变量 KBUILD_IMAGE 赋
值，如果已经赋值了则什么也不做。

#
# INSTALL_PATH specifies where to place the updated kernel and system map
# images. Default is /boot, but you can set it to other values
export INSTALL_PATH ?= /boot
# 用关键字声明了变量 INSTALL_PATH，使得该变量能传递到下级 Makefile 中。
# 符号“?=”在变量 INSTALL_PATH 没有赋值的情况下给变量 INSTALL_PATH 赋
值，如果已经赋值了则什么也不做。
```

```
#
# INSTALL_MOD_PATH specifies a prefix to MODLIB for module directory
# relocations required by build roots. This is not defined in the
# makefile but the argument can be passed to make if needed.
#

MODLIB = $(INSTALL_MOD_PATH)/lib/modules/$(KERNELRELEASE)
export MODLIB
# 给变量 MODLIB 赋值
# 用关键字声明变量 MODLIB 使得变量能够传到下级 Makefile

#
# INSTALL_MOD_STRIP, if defined, will cause modules to be
# stripped after they are installed. If INSTALL_MOD_STRIP is '1', then
# the default option --strip-debug will be used. Otherwise,
# INSTALL_MOD_STRIP will used as the options to the strip command.

ifdef INSTALL_MOD_STRIP
ifeq ($(INSTALL_MOD_STRIP),1)
mod_strip_cmd = $(STRIP) --strip-debug
else # 这个 else 与 ifeq ($(INSTALL_MOD_STRIP),1)对应。
mod_strip_cmd = $(STRIP) $(INSTALL_MOD_STRIP)
endif # INSTALL_MOD_STRIP=1
else # 这个 else 与 ifdef INSTALL_MOD_STRIP 对应。
mod_strip_cmd = true
endif # INSTALL_MOD_STRIP
export mod_strip_cmd
# 条件关键字 ifdef 判断变量 INSTALL_MOD_STRIP 是否被赋值，
# 条件关键字 ifeq 判断变量 INSTALL_MOD_STRIP 与 1 是否相同。
# 如果相同或被赋值则执行紧接着的命令，否则执行同一级的 else 后的命令。
# 上面命令就是在不同情况下给变量 mod_strip_cmd 赋值。

ifeq ($(KBUILD_EXTMOD),)
core-y     += kernel/ mm/ fs/ ipc/ security/ crypto/ block/
# 条件关键字 ifeq 判断变量 KBUILD_EXTMOD 的值与空格是否相同，如果是则为
变量 core-y 追加值。

vmlinux-dirs := $(patsubst %/,%,$(filter %/, $(init-y) $(init-m) \
$(core-y) $(core-m) $(drivers-y) $(drivers-m) \
$(net-y) $(net-m) $(libs-y) $(libs-m)))
# 将函数 patsubst 的返回值赋给变量 vmlinux-dirs。
# 函数 patsubst 的语法是:$(patsubst <pattern>,<replacement>,<text>)。
# 函数 patsubst 的作用是：用“%”替换函数 filter 返回的字符串中的“%”。
# 函数 filter 的作用是：
# 函数 filter 的作用是：返回字符串中符合模式“%/”的字符串。
```

```
vmlinux-alldirs := $(sort $(vmlinux-dirs) $(patsubst %/,%,$(filter %/,
\
$(init-n) $(init-) \
$(core-n) $(core-) $(drivers-n) $(drivers-) \
$(net-n) $(net-) $(libs-n)       $(libs-))))
# 将函数 sort 返回的值赋给变量 vmlinux-alldirs 。
# 函数 sort 的语法：$(sort <list>;)。
# 函数 sort 的返回值是：去掉相同的单词，给字符串<list>;中的单词排序（升序），返回排序后的字符串。
# 第一行中的 patsubst 是模式字符串替换函数，语法是：$(patsubst <pattern>;,<replacement>;,<text>;)。
# 在<text>中查找匹配模式”%/“的单词，并用”%“替换它们，最后返回替换后的字符串。

init-y     := $(patsubst %/, %/built-in.o, $(init-y))
core-y     := $(patsubst %/, %/built-in.o, $(core-y))
drivers-y := $(patsubst %/, %/built-in.o, $(drivers-y))
net-y     := $(patsubst %/, %/built-in.o, $(net-y))
libs-y1     := $(patsubst %/, %/lib.a, $(libs-y))
libs-y2     := $(patsubst %/, %/built-in.o, $(libs-y))
libs-y     := $(libs-y1) $(libs-y2)
# 上面的几个命令中，前几个是把函数 patsubst 返回值赋给左边的变量，最后一个直接把变量展开赋给左边的变量。
# 函数 patsubst 的语法是:$(patsubst <pattern>;,<replacement>;,<text>;)。
# 功能是：在字符串<text>;中查找与模式<pattern>;匹配的单词、替换它们，最后返回替换后的字符串。


# Build vmlinux
#
-------------------------------------------------------------------
------
# vmlinux is built from the objects selected by $(vmlinux-init) and
# $(vmlinux-main). Most are built-in.o files from top-level directories
# in the kernel tree, others are specified in arch/$(ARCH)/Makefile.
# Ordering when linking is important, and $(vmlinux-init) must be first.
#
# vmlinux
#     ^
#     |
#     +-< $(vmlinux-init)
#     |     +--< init/version.o + more
#     |
#     +--< $(vmlinux-main)
```

```
#     |       +--< driver/built-in.o mm/built-in.o + more
#     |
#     +-< kallsyms.o (see description in CONFIG_KALLSYMS section)
#
# vmlinux version (uname -v) cannot be updated during normal
# descending-into-subdirs phase since we do not yet know if we need to
# update vmlinux.
# Therefore this step is delayed until just before final link of vmlinux
-
# except in the kallsyms case where it is done just before adding the
# symbols to the kernel.
#
# System.map is generated to document addresses of all kernel symbols

vmlinux-init := $(head-y) $(init-y)
vmlinux-main := $(core-y) $(libs-y) $(drivers-y) $(net-y)
vmlinux-all := $(vmlinux-init) $(vmlinux-main)
vmlinux-lds := arch/$(ARCH)/kernel/vmlinux.lds
export KBUILD_VMLINUX_OBJS := $(vmlinux-all)
# 上面的命令都是给变量赋值。
# 最后一行命令中有关键词export,被export声明的变量能传到下级Makefile。

# Rule to link vmlinux - also used during CONFIG_KALLSYMS
# May be overridden by arch/$(ARCH)/Makefile
quiet_cmd_vmlinux__ ?= LD           $@
cmd_vmlinux__ ?= $(LD) $(LDFLAGS) $(LDFLAGS_vmlinux) -o $@ \
-T $(vmlinux-lds)
$(vmlinux-init)                                                    \
--start-group $(vmlinux-main)
--end-group                                        \
$(filter-out $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) FORCE ,$^)
# 赋值操作符"?="的作用是在他左边的变量还未赋值的情况下,把他右边的赋给
左边的变量。
# 反过滤函数filter-out的语法是：$(filter-out <pattern...>;,<text>;)
# 功能是:返回字符串$^中不符合模式(vmlinux-lds) $(vmlinux-init)
$(vmlinux-main) FORCE的字符串。

# Generate new vmlinux version
quiet_cmd_vmlinux_version = GEN         .version
cmd_vmlinux_version = set
-e;                                          \
if [ ! -r .version ]; then       \
rm -f .version;          \
echo 1 >.version;          \
else             \
```

```
mv .version .old_version;       \
expr 0$$(cat .old_version) + 1 >.version; \
fi;             \
$(MAKE) $(build)=init
# 上面是给两个变量 quiet_cmd_vmlinux_version、cmd_vmlinux_version 赋值。
# 其中 set、if...then...else... 、rm、echo、expr 等是 linux shell 中的命令。

# Generate System.map
quiet_cmd_sysmap = SYSMAP
cmd_sysmap = $(CONFIG_SHELL) $(srctree)/scripts/mksysmap
# 上面的是给变量 quiet_cmd_sysmap 和 cmd_sysmap 赋值。

# Link of vmlinux
# If CONFIG_KALLSYMS is set .version is already updated
# Generate System.map and verify that the content is consistent
# Use + in front of the vmlinux_version rule to silent warning with make -j2
# First command is ':' to allow us to use + in front of the rule
define rule_vmlinux__
:
$(if $(CONFIG_KALLSYMS),,+$(call cmd,vmlinux_version))

$(call cmd,vmlinux__)
$(Q)echo 'cmd_$@ := $(cmd_vmlinux__)' > $(@D)/.$(@F).cmd

$(Q)$(if
$($(quiet)cmd_sysmap),
                                \
echo ' $($(quiet)cmd_sysmap) System.map'
&&)                                             \
$(cmd_sysmap) $@
System.map;
                        \
if [ $$? -ne 0 ];
then
                                \
rm -f
$@;
                                        \
/bin/false;
                                                \
fi;
$(verify_kallsyms)
```

```
endef
# 关键词 define...endef 的作用是：为相同的命令序列定义一个变量，又称定义命令包。
# 上面定义了一个命令包,这个命令包的名字是(define 后的)rule_vmlinux__;
# 命令序列是：从 define 下面的那一行开始，直到 endef 的（包括）上面一行。
# 函数 if 的语法是：$(if <condition>;,<then-part>;,<else-part>;) 。
# 在 if 函数中，如果 CONFIG_KALLSYMS 的值为非空字符串，
# 那么执行<then-part>;部分（这里为空即第一二个逗号之间的部分）；
# 如果 CONFIG_KALLSYMS 的值为空字符串（即为假），则执行+$(call cmd,vmlinux_version)。
# 函数 call 的语法是:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
# 功能是：用参数<parm1>;,<parm2>;,<parm3>;...依次取代参数<expression>中的变量，
# 并返回取代后的<expression>。
# echo 是 linux shell 中的一个显示文字的命令，语法是：：echo [-ne][字符串]或 echo [--help][--version]
# rm 是 linux shell 中的一个删除文件、文件夹的命令。

ifdef CONFIG_KALLSYMS
# 条件关键词 ifdef 判断变量 CONFIG_KALLSYMS 是否非空。
# Generate section listing all symbols and add it into vmlinux $(kallsyms.o)
# It's a three stage process:
# o .tmp_vmlinux1 has all symbols and sections, but __kallsyms is
#     empty
#     Running kallsyms on that gives us .tmp_kallsyms1.o with
#     the right size - vmlinux version (uname -v) is updated during this step
# o .tmp_vmlinux2 now has a __kallsyms section of the right size,
#     but due to the added section, some addresses have shifted.
#     From here, we generate a correct .tmp_kallsyms2.o
# o The correct .tmp_kallsyms2.o is linked into the final vmlinux.
# o Verify that the System.map from vmlinux matches the map from
#     .tmp_vmlinux2, just in case we did not generate kallsyms correctly.
# o If CONFIG_KALLSYMS_EXTRA_PASS is set, do an extra pass using
#     .tmp_vmlinux3 and .tmp_kallsyms3.o. This is only meant as a
#     temporary bypass to allow the kernel to be built while the
#     maintainers work out what went wrong with kallsyms.

ifdef CONFIG_KALLSYMS_EXTRA_PASS
last_kallsyms := 3
else     # 这个 else 与 ifdef CONFIG_KALLSYMS_EXTRA_PASS 对应
last_kallsyms := 2
endif     # 这个 endif 与 ifdef CONFIG_KALLSYMS_EXTRA_PASS 对应
```

嵌套了一个 ifdef。
如果变量 CONFIG_KALLSYMS_EXTRA_PASS 是非空，则执行 else 前面的部分，否则执行 else 到 endif 之间的部分。

kallsyms.o := .tmp_kallsyms$(last_kallsyms).o
为变量 kallsyms.o 赋值。

```
define verify_kallsyms
$(Q)$(if
$($(quiet)cmd_sysmap),
                                \
echo ' $($(quiet)cmd_sysmap) .tmp_System.map'
&&)                                     \
$(cmd_sysmap) .tmp_vmlinux$(last_kallsyms) .tmp_System.map
$(Q)cmp -s System.map .tmp_System.map
||                                                              \
(echo Inconsistent kallsyms
data;                                                           \
echo Try setting
CONFIG_KALLSYMS_EXTRA_PASS;                                     \
rm .tmp_kallsyms* ; /bin/false )
endef
```

关键词 define 定义了一个叫 verify_kallsyms 的命令包。
定义这种命令序列的语法以“define”开始，以“endef”结束。
函数 if 的语法是：$(if <condition>;,<then-part>;,<else-part>;)。
如果($(quiet)cmd_sysmap)，返回为非空字符串，那么执行 echo
cmp 是 linux 中的命令，语法是：cmp [-clsv][-i <字符数目>][--help][第一个文件][第二个文件]
cmp 的功能是：比较两个文件是否有差异。
echo 也是 linux 中的命令，用于显示字符串。语法是：echo [-ne][字符串] 或 echo [--help][--version]
rm 也是 linux 中的命令，语法是：rm [选项]... 目录... 作用是：删除指定的<文件>(即解除链接)。

```
# Update vmlinux version before link
# Use + in front of this rule to silent warning about make -j1
# First command is ':' to allow us to use + in front of this rule
cmd_ksym_ld = $(cmd_vmlinux__)
define rule_ksym_ld
:
+$(call cmd,vmlinux_version)
$(call cmd,vmlinux__)
$(Q)echo 'cmd_$@ := $(cmd_vmlinux__)' > $(@D)/.$(@F).cmd
endef
```

为变量 cmd_ksym_ld 赋值。

# 关键词 define 定义了名叫 rule_ksym_ld 的命令包。
# 命令包 rule_ksym_ld 里面的命令序列是 define 下面一行开始到 endef 上一行结束。
# 函数 call 的语法是:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
# 函数 call 的功能是：<expression>;参数中的变量，被参数<parm1>;，<parm2>;，<parm3>;依次取代。
# 函数 call 的返回值是被取代后的<expression>。
# echo 是 linux 中的命令，它的语法是：echo [-ne][字符串]或 echo [--help][--version]
# echo 的功能是：echo 会将输入的字符串送往标准输出。输出的字符串间以空白字符隔开，并在最后加上换行号。
# 符号>用来改变送出的数据信道(stdout, stderr)，使之输出到指定的档案。这里是输出到$(@D)/.$(@F).cmd。
# Generate .S file with all kernel symbols
quiet_cmd_kallsyms = KSYM $@
cmd_kallsyms = $(NM) -n $< | $(KALLSYMS) \
$(if $(CONFIG_KALLSYMS_ALL),--all-symbols) > $@
分别为变量 quiet_cmd_kallsyms 和 cmd_kallsyms 赋值。
自动变量“$@”表示表示规则中的目标文件集，自动变量“$<”表示依赖目标中的第一个目标名字。
函数 if 的语法是；$(if <condition>;,<then-part>;,<else-part>;)，这里没有<then-part>。
如果<condition>为非空字符串，于是<then-part>;会被计算。
符号>是 linux 中的重定向标志。用来改变送出的数据信道(stdout, stderr)，使之输出到指定的档案；
符号|是 linux 中的管道标志。功能是：上一个命令的 stdout 接到下一个命令的 stdin。

.tmp_kallsyms1.o .tmp_kallsyms2.o .tmp_kallsyms3.o: %.o: %.S scripts FORCE
$(call if_changed_dep,as_o_S)
定义了一个多目标规则。
函数 call 的语法是:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
功能是：参数<expression>中的变量，被参数<parm1>;，<parm2>;，<parm3>;依次取代
返回值是：被替换后的<expression>。

.tmp_kallsyms%.S: .tmp_vmlinux% $(KALLSYMS)
$(call cmd,kallsyms)
定义了一个规则，目标为.tmp_kallsyms%.S，依赖文件为.tmp_vmlinux% $(KALLSYMS)。
函数 call 的语法是:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
功能是：参数<expression>中的变量，被参数<parm1>;，<parm2>;，<parm3>;

依次取代
返回值是：被替换后的<expression>。

.tmp_vmlinux1 must be complete except kallsyms, so update vmlinux version
.tmp_vmlinux1: $(vmlinux-lds) $(vmlinux-all) FORCE
$(call if_changed_rule,ksym_ld)
定义了一个规则，目标为.tmp_vmlinux1，依赖文件为$(vmlinux-lds) $(vmlinux-all)。
函数call的语法是:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
功能是：参数<expression>中的变量，被参数<parm1>;，<parm2>;，<parm3>;依次取代
返回值是：被替换后的<expression>。

.tmp_vmlinux2: $(vmlinux-lds) $(vmlinux-all) .tmp_kallsyms1.o FORCE
$(call if_changed,vmlinux__)
定义了一个规则，目标是.tmp_vmlinux2。
函数call的语法是:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
功能是：参数<expression>中的变量，被参数<parm1>;，<parm2>;，<parm3>;依次取代
返回值是：被替换后的<expression>。

.tmp_vmlinux3: $(vmlinux-lds) $(vmlinux-all) .tmp_kallsyms2.o FORCE
$(call if_changed,vmlinux__)
定义了一个规则，目标是.tmp_vmlinux3
函数call的语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
功能是：参数<expression>中的变量，被参数<parm1>;，<parm2>;，<parm3>;依次取代
返回值是：被替换后的<expression>。

Needs to visit scripts/ before $(KALLSYMS) can be used.
$(KALLSYMS): scripts ;
定义了一个（分号前面没有命令）没有命令的规则，

Generate some data for debugging strange kallsyms problems
debug_kallsyms: .tmp_map$(last_kallsyms)
定义了一个依赖关系，目标为debug_kallsyms，依赖文件为 .tmp_map$(last_kallsyms)

.tmp_map%: .tmp_vmlinux% FORCE
($(OBJDUMP) -h $< | $(AWK) '/^ +[0-9]/{print $$4 " 0 " $$2}'; $(NM) $<) | sort > $@
定义了一个规则，目标为.tmp_map%，依赖文件为.tmp_vmlinux%
自动化变量$<表示依赖目标中的目标文件集。自动化变量$@表示依赖目标中的

第一个目标名字。
符号“|”表示管道，作用是：使上一个命令的 stdout 接到下一个命令的 stdin。
符号“>”用来改变送出的数据信道，使之输出到指定的档案（文件等）。

.tmp_map3: .tmp_map2
定义了一个依赖关系，目标是.tmp_map3，依赖文件是.tmp_map2。

.tmp_map2: .tmp_map2
定义了一个依赖关系，目标是.tmp_map2，依赖文件是.tmp_map2。

endif # ifdef CONFIG_KALLSYMS
这个 endif 与 ifdef CONFIG_KALLSYMS 对应

.tmp_map3: .tmp_map2
定义了一个依赖关系，目标是.tmp_map3，依赖文件是.tmp_map2。

.tmp_map2: .tmp_map2
定义了一个依赖关系，目标是.tmp_map2，依赖文件是.tmp_map2。

endif # ifdef CONFIG_KALLSYMS
这个 endif 与 ifdef CONFIG_KALLSYMS 对应

vmlinux image - including updated kernel symbols
vmlinux: $(vmlinux-lds) $(vmlinux-init) $(vmlinux-main) $(kallsyms.o) FORCE
ifdef CONFIG_HEADERS_CHECK
(Q)(MAKE) -f $(srctree)/Makefile headers_check
endif
$(call if_changed_rule,vmlinux__)
(Q)(MAKE) -f $(srctree)/scripts/Makefile.modpost $@
$(Q)rm -f .old_version
定义了一个规则，目标是 vmlinux
条件关键词 ifdef 的语法是：ifdef <variable-name>;
如果变量<variable-name>;的值非空，那到表达式为真。否则，表达式为假。
函数 call 的语法是:$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
函数 call 的功能是：<expression>;参数中的变量，被参数<parm1>;，<parm2>;，<parm3>;依次取代。
rm 是 linux 中的一个命令，-f 是命令 rm 的参数，.old_version 是要删除的文件夹。

The actual objects are generated when descending,
make sure no implicit rule kicks in
$(sort $(vmlinux-init) $(vmlinux-main)) $(vmlinux-lds):

```
$(vmlinux-dirs) ;
# 排序函数 sort 的语法是：$(sort <list>;),
# 功能是：去掉<list>;中相同的单词，然后给字符串<list>;中剩下的单词排序（升序）。返回排好的字符串。

# Handle descending into subdirectories listed in $(vmlinux-dirs)
# Preset locale variables to speed up the build process. Limit locale
# tweaks to this spot to avoid wrong language settings when running
# make menuconfig etc.
# Error messages still appears in the original language

PHONY += $(vmlinux-dirs)
vmlinux-dirs: prepare scripts
$(Q)$(MAKE) $(build)=$@
# 为变量 PHONY 追加变量 vmlinux-dirs 的值。
# 第二行定义了一个规则，目标是 vmlinux-dirs，依赖文件是 prepare 和 scripts。

# Build the kernel release string
#
# The KERNELRELEASE value built here is stored in the file
# include/config/kernel.release, and is used when executing several
# make targets, such as "make install" or "make modules_install."
#
# The eventual kernel release string consists of the following fields,
# shown in a hierarchical format to show how smaller parts are concatenated
# to form the larger and final value, with values coming from places like
# the Makefile, kernel config options, make command line options and/or
# SCM tag information.
#
# $(KERNELVERSION)
#     $(VERSION)       eg, 2
#     $(PATCHLEVEL)       eg, 6
#     $(SUBLEVEL)       eg, 18
#     $(EXTRAVERSION)     eg, -rc6
# $(localver-full)
#     $(localver)
#         localversion*     (files without backups, containing '~')
#         $(CONFIG_LOCALVERSION) (from kernel config setting)
#     $(localver-auto)     (only if CONFIG_LOCALVERSION_AUTO is set)
#         ./scripts/setlocalversion (SCM tag, if one exists)
#         $(LOCALVERSION)     (from make command line if provided)
#
# Note how the final $(localver-auto) string is included *only* if the
# kernel config option CONFIG_LOCALVERSION_AUTO is selected. Also, at the
```

```
# moment, only git is supported but other SCMs can edit the script
# scripts/setlocalversion and add the appropriate checks as needed.

pattern = ".*/localversion[^~]*"
string = $(shell cat /dev/null \
`find $(objtree) $(srctree) -maxdepth 1 -regex $(pattern) | sort -u`)
# 为变量 pattern 和 string 赋值
# 函数 shell 新生成一个 shell 程序来执行 shell 函数后面的命令(如 cat、find 等)
# cat 是 linux 中的一个命令，语法是 cat filename ，功能是一次显示整个文件。

localver = $(subst $(space),, $(string) \
$(patsubst "%",%,$(CONFIG_LOCALVERSION)))
# 给变量 localver 赋值。
# subst 是 Makefile 的一个字符串替换函数，语法是：$(subst <from>;,<to>;,<text>;)。
# 功能是：把字串<text>;中的<from>;字符串替换成<to>;，然后返回被替换过后的字符串。

# If CONFIG_LOCALVERSION_AUTO is set scripts/setlocalversion is called
# and if the SCM is know a tag from the SCM is appended.
# The appended tag is determined by the SCM used.
#
# Currently, only git is supported.
# Other SCMs can edit scripts/setlocalversion and add the appropriate
# checks as needed.
ifdef CONFIG_LOCALVERSION_AUTO
_localver-auto = $(shell $(CONFIG_SHELL) \
$(srctree)/scripts/setlocalversion $(srctree))
localver-auto = $(LOCALVERSION)$(_localver-auto)
endif
# ifdef 是 Makefile 的一种条件关键字，语法是：ifdef <variable-name>;
# 功能是：如果变量<variable-name>;的值非空，那到表达式为真。否则，表达式为假。
# 如果变量<variable-name>;的值非空，则为变量 _localver-auto 和 localver-auto 赋值。

localver-full = $(localver)$(localver-auto)
# 为变量 localver-full 赋值。

# Store (new) KERNELRELASE string in include/config/kernel.release
kernelrelease = $(KERNELVERSION)$(localver-full)
include/config/kernel.release: include/config/auto.conf FORCE
$(Q)rm -f $@
```

```
$(Q)echo $(kernelrelease) > $@
# 为变量 kernelrelease 赋值。
# 定义了一个规则，目标为 include/config/kernel.release，依赖文件为 include/config/auto.conf FORCE。
# rm 是 linux 中的一个命令，语法是：rm [选项]... 目录... ，功能是：目录... 删除指定的文件或目录
# echo 是 linux 下的一个命令，语法是：echo 字符串，功能是：将字符串送往标准输出。


# Things we need to do before we recursively start building the kernel
# or the modules are listed in "prepare".
# A multi level approach is used. prepareN is processed before prepareN-1.
# archprepare is used in arch Makefiles and when processed asm symlink,
# version.h and scripts_basic is processed / created.

# Listed in dependency order
PHONY += prepare archprepare prepare0 prepare1 prepare2 prepare3
# 给变量 PHONY 追加值

# prepare3 is used to check if we are building in a separate output directory,
# and if so do:
# 1) Check that make has not been executed in the kernel src $(srctree)
# 2) Create the include2 directory, used for the second asm symlink
prepare3: include/config/kernel.release
ifneq ($(KBUILD_SRC),)
@echo '  Using $(srctree) as source for kernel'
$(Q)if [ -f $(srctree)/.config -o -d $(srctree)/include/config ]; then \
echo "  $(srctree) is not clean, please run 'make mrproper'";\
echo "  in the '$(srctree)' directory.";\
/bin/false; \
fi;
$(Q)if [ ! -d include2 ]; then mkdir -p include2; fi;
$(Q)ln -fsn $(srctree)/include/asm-$(ARCH) include2/asm
endif
# 定义了一个规则。目标是 prepare3，依赖文件是 include/config/kernel.release
# ifneq 是 Makefile 的一个条件关键词，语法是 ifneq (<arg1>;, <arg2>;)
# 功能是:比较参数“arg1”和“arg2”的值是否相同，如果不同，则为真，否则为假。

# prepare2 creates a makefile if using a separate output directory
prepare2: prepare3 outputmakefile
```

```
# 定义了一个依赖关系。目标为 prepare2，依赖文件是：prepare3 和 outputmakefile

prepare1: prepare2 include/linux/version.h include/linux/utsrelease.h \
include/asm include/config/auto.conf
ifneq ($(KBUILD_MODULES),)
$(Q)mkdir -p $(MODVERDIR)
$(Q)rm -f $(MODVERDIR)/*
endif
# 定义了一个规则，目标为 prepare1。
# ifneq 是 Makefile 的一个条件关键词，其语法是：ifneq (<arg1>;, <arg2>;)
# 功能是:比较参数“arg1”和“arg2”的值是否相同，如果不同，则为真，否则为假。
# rm 是 linux shell 中的一个用于删除文件或文件夹的命令。
# 命令 rm 的语法是：rm [选项]... 文件...

archprepare: prepare1 scripts_basic
# 定义了一个依赖关系，目标是 archprepare，依赖文件是 prepare1 scripts_basic。

prepare0: archprepare FORCE
$(Q)$(MAKE) $(build)=.
$(Q)$(MAKE) $(build)=. missing-syscalls
# 定义了一个规则，目标是 prepare0，依赖文件是 archprepare FORCE

# All the preparing..
prepare: prepare0
# 定义了一个依赖关系，目标是 prepare，依赖文件是 prepare0

# Leave this as default for preprocessing vmlinux.lds.S, which is now
# done in arch/$(ARCH)/kernel/Makefile

export CPPFLAGS_vmlinux.lds += -P -C -U$(ARCH)
# 为变量 CPPFLAGS_vmlinux.lds 追加值。
# export 是 Makefile 的一个关键词，用来声明一个变量，使这个变量能传到下级 Makefile
# 语法是：export <variable ...>;

# FIXME: The asm symlink changes when $(ARCH) changes. That's
# hard to detect, but I suppose "make mrproper" is a good idea
# before switching between archs anyway.

include/asm:
@echo '  SYMLINK $@ -> include/asm-$(ARCH)'
$(Q)if [ ! -d include ]; then mkdir -p include; fi;
```

```
@ln -fsn asm-$(ARCH) $@
# 定义了一个伪目标 include/asm
# 如果字符“@”在命令行前，那么这个命令将不被 make 显示出来。
# 通常 make 会把其要执行的命令行在命令执行前输出到屏幕上。
# if 是 Makefile 的一个函数，其语法是：$(if <condition>;,<then-part>;,<else-part>;)
# 功能是：。<condition>;参数是 if 的表达式，如果其返回的为非空字符串，
# 那么这个表达式就相当于返回真，于是，<then-part>;会被计算，否则<else-part>;会被计算。
# ln 是 linux shell 中的一个命令，其语法是：ln [ -f | -n] [ -s ] SourceFile [ TargetFile ]
# 功能是:在 SourceFile 参数中指定的文件链接到在 TargetFile 参数中指定的文件。

# Generate some files
#
------------------------------------------------------------------------
------

# KERNELRELEASE can change from a few different places, meaning version.h
# needs to be updated, so this check is forced on all builds

uts_len := 64
define filechk_utsrelease.h
if [ `echo -n "$(KERNELRELEASE)" | wc -c ` -gt $(uts_len) ]; then \
echo '"$(KERNELRELEASE)" exceeds $(uts_len) characters' >&2;       \
exit
1;
                                                    \
fi;
                                                                  \
(echo \#define UTS_RELEASE \"$(KERNELRELEASE)\";)
endef
# 第一行为变量 uts_len 赋值。
# define 是 Makefile 的一个关键字，用于定义命令包。此处定义了 filechk_utsrelease.h 命令包。
# echo 是 linux shell 中的一个命令，它的语法是：echo [字符串]，功能是；将字符串送往标准输出。
# 字符”|“是 linux shell 中的管道标志，语法是：命令 | 命令，
# 功能是：前一个命令的输出重定向到后一个命令的输入
# 字符”\“是续行的标志，由于某些命令要建立在前面的命令的基础之上，
# 所以必须把它与前面的命令写在同一行上，这时就要用到字符”\“来续行。

define filechk_version.h
(echo                   \#define                   LINUX_VERSION_CODE
```

```
$(shell                                                    \
expr $(VERSION) \* 65536 + $(PATCHLEVEL) \* 256 + $(SUBLEVEL));          \
echo '#define KERNEL_VERSION(a,b,c) (((a) << 16) + ((b) << 8) + (c))';)
endef
# export 是 Makefile 的一个关键字，用来定义命令包（相同的命令序列定义一个变量，即命令包）。
# echo 是 linux shell 中的一个命令，它的语法是：echo [字符串]，功能是；将字符串送往标准输出。
# 字符”\“是续行的标志，由于某些命令要建立在前面的命令的基础之上，
# 所以必须把它与前面的命令写在同一行上，这时就要用到字符”\“来续行。
# expr 是 linux shell 中的一个字符串处理命令。

include/linux/version.h: $(srctree)/Makefile FORCE
$(call filechk,version.h)
# 定义了一个规则，目标是 include/linux/version.h，依赖文件是 $(srctree)/Makefile 和 FORCE
# call 是 Makefile 中的一个命令，其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
# 功能是:<expression>参数中的变量,如$(1),$(2),$(3)等,会被参数<parm1>,<parm2>,<parm3>依次取代。

include/linux/utsrelease.h: include/config/kernel.release FORCE
$(call filechk,utsrelease.h)
# 定义了一个规则，目标是 include/linux/utsrelease.h，依赖文件是 include/config/kernel.release 和 FORCE
# call 是 Makefile 中的一个命令，其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
# 功能是：<expression>;参数中的变量，如$(1)，$(2)，$(3)等，会被参数<parm1>,<parm2>,<parm3>依次取代。

#
-------------------------------------------------------------------
------

PHONY += depend dep
depend dep:
@echo '*** Warning: make $@ is unnecessary now.'
# 第一行为变量 PHONY 追加值 depend 和 dep，第二行定义了伪目标 depend dep。
# echo 是 linux shell 中的一个命令，它的语法是：echo [字符串]，功能是；将字符串送往标准输出。

#
-------------------------------------------------------------------
------
# Kernel headers
```

```
INSTALL_HDR_PATH=$(objtree)/usr
export INSTALL_HDR_PATH
# 第一行为变量 INSTALL_HDR_PATH 赋值。
# export 是 Makefile 的一个关键字,用来声明变量,使变量能传到下级 Makefile

HDRARCHES=$(filter-out                                    generic,$(patsubst
$(srctree)/include/asm-%/Kbuild,%,$(wildcard
$(srctree)/include/asm-*/Kbuild)))
# 为变量 HDRARCHES 赋值。
# filter-out 是 Makefile 的一个字串处理函数。其语法是：$(filter-out
<pattern...>;,<text>;)
# 功能是以<pattern>;模式过滤<text>;字符串中的单词，去除符合模式
<pattern>;的单词。可以有多个模式。
# patsubst 是 Makefile 中的模式字符串替换函数，其语法是：$(patsubst
<pattern>;,<replacement>;,<text>;)
# 功能是：查找<text>;中的单词是否符合模式<pattern>;，如果匹配的话，则
以<replacement>;替换。
# wildcard 是 Makefile 的一个关键字，在给变量赋值时，使通配符在变量中展
开。

PHONY += headers_install_all
headers_install_all: include/linux/version.h scripts_basic FORCE
$(Q)$(MAKE) $(build)=scripts scripts/unifdef
$(Q)for arch in $(HDRARCHES); do \
$(MAKE)    ARCH=$$arch    -f    $(srctree)/scripts/Makefile.headersinst
obj=include BIASMDIR=-bi-$$arch ;\
done
# 第一行为变量 PHONY 追加值
# 第二行定义了一个规则，目标是 headers_install_all。

PHONY += headers_install
headers_install: include/linux/version.h scripts_basic FORCE
@if [ ! -r $(srctree)/include/asm-$(ARCH)/Kbuild ]; then \
echo '*** Error: Headers not exportable for this architecture ($(ARCH))';
\
exit 1 ; fi
$(Q)$(MAKE) $(build)=scripts scripts/unifdef
$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.headersinst obj=include
# 第一行为变量 PHONY 追加值
# 第二行定义了一个规则，目标是 headers_install。

PHONY += headers_check_all
headers_check_all: headers_install_all
$(Q)for arch in $(HDRARCHES); do \
$(MAKE)    ARCH=$$arch    -f    $(srctree)/scripts/Makefile.headersinst
```

```
obj=include BIASMDIR=-bi-$$arch HDRCHECK=1 ;\
done
# 第一行为变量 PHONY 追加值
# 第二行定义了一个规则，目标是 headers_check_all。

PHONY += headers_check
headers_check: headers_install
$(Q)$(MAKE)  -f  $(srctree)/scripts/Makefile.headersinst  obj=include
HDRCHECK=1
# 第一行为变量 PHONY 追加值
# 第二行定义了一个规则，目标是 headers_check。
#
---------------------------------------------------------------------
------
# Modules

ifdef CONFIG_MODULES
# ifdef 是 Makefile 中的一个条件关键词，其语法是：ifdef <variable-name>;
# 如果变量<variable-name>;的值非空，那到表达式为真。否则，表达式为假。

# By default, build modules as well

all: modules
# 定义了一个依赖关系，目标是 all，依赖文件是 modules

# Build modules

PHONY += modules
modules: $(vmlinux-dirs) $(if $(KBUILD_BUILTIN),vmlinux)
@echo ' Building modules, stage 2.';
$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost
# 为变量 PHONY 追加值
# 定义了一个规则，目标是 modules。
# if 是 Makefile 中的一个函数，其语法是：$(if
<condition>;,<then-part>;,<else-part>;)
# <condition>;参数是 if 的表达式，如果其返回的为非空字符串，那么这个表
达式就相当于返回真，
# 于是，<then-part>;会被计算，否则<else-part>;会被计算。
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。


# Target to prepare building external modules
PHONY += modules_prepare
modules_prepare: prepare scripts
# 为变量 PHONY 追加值
```

```
# 第二行定义了依赖关系，目标是 modules_prepare，依赖文件是 prepare
scripts

# Target to install modules
PHONY += modules_install
modules_install: _modinst_ _modinst_post
# 第一行为变量追加值。
# 第二行定义了依赖关系，目标是 modules_install，依赖文件是_modinst_
_modinst_post

PHONY += _modinst_
_modinst_:
@if [ -z "`$(DEPMOD) -V 2>/dev/null | grep module-init-tools`" ]; then
\
echo "Warning: you may need to install module-init-tools"; \
echo "See http://www.codemonkey.org.uk/docs/post-halloween-2.6.txt";\
sleep 1; \
fi
@rm -rf $(MODLIB)/kernel
@rm -f $(MODLIB)/source
@mkdir -p $(MODLIB)/kernel
@ln -s $(srctree) $(MODLIB)/source
@if [ ! $(objtree) -ef $(MODLIB)/build ]; then \
rm -f $(MODLIB)/build ; \
ln -s $(objtree) $(MODLIB)/build ; \
fi
$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modinst
# 第一行为变量 PHONY 追加值。
# 第二行定义了伪目标_modinst_。
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
# rm 是 linux shell 中的用于删除文件或目录的命令。其语法是：rm [选项]...
[要删除的文件]...
# mkdir 是 linux shell 中的用于创建目录的命令，其语法是：mkdir [OPTION]
DIRECTORY...
# ln 是 linux shell 中的用于创建一个链接，其语法是：ln [OPTION]... [-T]
TARGET LINK_NAME

# If System.map exists, run depmod. This deliberately does not have a
# dependency on System.map since that would run the dependency tree on
# vmlinux. This depmod is only for convenience to give the initial
# boot a modules.dep even before / is mounted read-write. However the
# boot script depmod is the master version.
ifeq "$(strip $(INSTALL_MOD_PATH))" ""
depmod_opts :=
else
```

```
depmod_opts := -b $(INSTALL_MOD_PATH) -r
endif
PHONY += _modinst_post
_modinst_post: _modinst_
if [ -r System.map -a -x $(DEPMOD) ]; then $(DEPMOD) -ae -F System.map
$(depmod_opts) $(KERNELRELEASE); fi
# ifeq 是 Makefile 中的一个条件关键词，其语法是：ifeq "<arg1>;""<arg2>;"
# strip 是 Makefile 中的用于去掉空格的函数。其语法是：$(strip <string>;)
# 第二四行分别在不同的情况下给变量 depmod_opts 赋值。
# 第六行给变量 PHONY 追加值。
# 第七行定义了一个规则，目标是_modinst_post，依赖文件是_modinst_

else # CONFIG_MODULES
# 上面这个 else 与 ifdef CONFIG_MODULES 对应。

# Modules not configured
#
-------------------------------------------------------------------
------

modules modules_install: FORCE
@echo
@echo "The present kernel configuration has modules disabled."
@echo "Type 'make config' and enable loadable module support."
@echo "Then build a kernel with module support enabled."
@echo
@exit 1
# 定义了一个规则，目标是 modules modules_install，依赖文件是 FORCE
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
# echo 是 Makefile 中的一个将输入的字符串送往标准输出关键字，其语法是：
echo [-ne][字符串]
# exit 是 linux shell 中的一个退出目前的 shell 的命令，其语法是：exit [状
态值]

endif # CONFIG_MODULES
# 上面这个 endif 与 ifdef CONFIG_MODULES 对应。

###
# Cleaning is done on three levels.
# make clean         Delete most generated files
#                              Leave enough to build external modules
# make mrproper Delete the current configuration, and all generated files
# make distclean Remove editor backup files, patch leftover files and the
like
```

```
# Directories & files removed with 'make clean'
CLEAN_DIRS += $(MODVERDIR)
CLEAN_FILES += vmlinux System.map \
.tmp_kallsyms* .tmp_version .tmp_vmlinux* .tmp_System.map
# 第一行将变量 MODVERDIR 的值追加给变量 CLEAN_DIRS
# 第二行给变量 CLEAN_FILES 追加值。

# Directories & files removed with 'make mrproper'
MRPROPER_DIRS += include/config include2 usr/include
MRPROPER_FILES += .config .config.old include/asm .version .old_version
\
include/linux/autoconf.h include/linux/version.h           \
include/linux/utsrelease.h
             \
Module.symvers tags TAGS cscope*
# 上面的是给变量 MRPROPER_DIRS 和 MRPROPER_FILES 追加值，
# 字符“\”有两个功能：一是：续行；二是：转义。

# clean - Delete most, but leave enough to build external modules
#
clean: rm-dirs := $(CLEAN_DIRS)
clean: rm-files := $(CLEAN_FILES)
clean-dirs                :=      $(addprefix      _clean_,$(srctree)
$(vmlinux-alldirs))
# 第一二行分别定义了一个规则，每个规则的命令都是给依赖文件赋值。
# 第三行定义了一个依赖关系。

PHONY += $(clean-dirs) clean archclean
$(clean-dirs):
$(Q)$(MAKE) $(clean)=$(patsubst _clean_%,%,$@)
# 第一行给变量 PHONY 追加值。
# 第二行定义了一个伪目标$(clean-dirs)。

clean: archclean $(clean-dirs)
$(call cmd,rmdirs)
$(call cmd,rmfiles)
@find . $(RCS_FIND_IGNORE) \
\( -name '*.[oas]' -o -name '*.ko' -o -name '.*.cmd' \
-o -name '.*.d' -o -name '.*.tmp' -o -name '*.mod.c' \
-o -name '*.symtypes' \) \
-type f -print | xargs rm -f
# 第一行定义了一个规则,目标是 clean,依赖文件是 archclean 和$(clean-dirs)
# call 是 Makefile 的一个用来创建新的参数化的函数，
# 其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
# <expression>;参数中的变量，会被参数<parm1>;，<parm2>;，<parm3>;依次
```

取代。
find 是 linux shell 中的一个查找命令，其语法是：find path option [-print -exec -ok]
当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
字符“\”有两个功能：一是：续行；二是：转义。

```
# mrproper - Delete all generated files, including .config
#
mrproper: rm-dirs := $(wildcard $(MRPROPER_DIRS))
mrproper: rm-files := $(wildcard $(MRPROPER_FILES))
mrproper-dirs                    :=                $(addprefix
_mrproper_,Documentation/DocBook scripts)
```

第一二行分别定义了一个规则，它们的命令分别是给依赖文件赋值。
第三行给变量 mrproper-dirs 赋值。

```
PHONY += $(mrproper-dirs) mrproper archmrproper
$(mrproper-dirs):
$(Q)$(MAKE) $(clean)=$(patsubst _mrproper_%,%,$@)
```

第一行给变量 PHONY 赋值。
第二行定义了一个伪目标$(mrproper-dirs)。
patsubst 是 Makefile 中的一个模式字符串替换函数，
其语法是：$(patsubst <pattern>;,<replacement>;,<text>;)
查找<text>;中的单词是否符合模式<pattern>;，如果匹配的话，则以<replacement>;替换。

```
mrproper: clean archmrproper $(mrproper-dirs)
$(call cmd,rmdirs)
$(call cmd,rmfiles)
```

定义了一个规则，目标是 mrproper。
call 是 Makefile 的一个用来创建新的参数化的函数，
其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
<expression>;参数中的变量，会被参数<parm1>;，<parm2>;，<parm3>;依次取代。

```
# distclean
#
PHONY += distclean
```

给变量 PHONY 追加值

```
distclean: mrproper
@find $(srctree) $(RCS_FIND_IGNORE) \
\( -name '*.orig' -o -name '*.rej' -o -name '*~' \
-o -name '*.bak' -o -name '#*#' -o -name '.*.orig' \
-o -name '.*.rej' -o -size 0 \
-o -name '*%' -o -name '.*.cmd' -o -name 'core' \) \
```

```
-type f -print | xargs rm -f
# 上面定义了一个规则，目标是 distclean，依赖文件是 mrproper。
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
# find 是 linux shell 中的一个查找命令，其语法是：find path option [-print
-exec -ok]
# 字符”\“有两个用途：一是：转义；二是：续行。

# Packaging of the kernel to various formats
#
------------------------------------------------------------------------
------
# rpm target kept for backward compatibility
package-dir := $(srctree)/scripts/package

%pkg: include/config/kernel.release FORCE
$(Q)$(MAKE) $(build)=$(package-dir) $@
rpm: include/config/kernel.release FORCE
$(Q)$(MAKE) $(build)=$(package-dir) $@
# 上面定义了两个规则，目标分别是%pkg 和 rpm。
@ $@是 Makefile 的自动化变量，表示当前规则的目标集。

# Brief documentation of the typical targets used
#
------------------------------------------------------------------------
------

boards := $(wildcard $(srctree)/arch/$(ARCH)/configs/*_defconfig)
boards := $(notdir $(boards))
# 上面两行分别给变量 boards 赋值。
# wildcard 是 Makefile 中的关键字，用于让通配符在变量中展开。
# notdir 是 Makefile 中的取文件函数，其语法是：$(notdir <names...>;)
# 功能：从文件名序列<names>;中取出非目录部分。非目录部分是指最后一个反
斜杠（“/”）之后的部分。

help:
@echo 'Cleaning targets:'
@echo ' clean       - Remove most generated files but keep the config and'
@echo '                                         enough build support to
build external modules'
@echo ' mrproper     - Remove all generated files + config + various
backup files'
@echo ' distclean     - mrproper + remove editor backup and patch files'
@echo ''
@echo 'Configuration targets:'
@$(MAKE) -f $(srctree)/scripts/kconfig/Makefile help
```

```
@echo ''
@echo 'Other generic targets:'
@echo ' all       - Build all targets marked with [*]'
@echo '* vmlinux     - Build the bare kernel'
@echo '* modules     - Build all modules'
@echo '  modules_install - Install all modules to INSTALL_MOD_PATH
(default: /)'
@echo ' dir/                        - Build all files in dir and below'
@echo ' dir/file.[ois] - Build specified target only'
@echo ' dir/file.ko         - Build module including final link'
@echo ' rpm       - Build a kernel as an RPM package'
@echo ' tags/TAGS     - Generate tags file for editors'
@echo ' cscope     - Generate cscope index'
@echo ' kernelrelease     - Output the release version string'
@echo ' kernelversion     - Output the version stored in Makefile'
@if [ -r $(srctree)/include/asm-$(ARCH)/Kbuild ]; then \
echo '  headers_install  -  Install  sanitised  kernel  headers  to
INSTALL_HDR_PATH'; \
echo '                                                  (default:
$(INSTALL_HDR_PATH))'; \
fi
@echo ''
@echo 'Static analysers'
@echo ' checkstack           - Generate a list of stack hogs'
@echo ' namespacecheck - Name space analysis on compiled kernel'
@if [ -r $(srctree)/include/asm-$(ARCH)/Kbuild ]; then \
echo ' headers_check     - Sanity check on exported headers'; \
fi
@echo ''
@echo 'Kernel packaging:'
@$(MAKE) $(build)=$(package-dir) help
@echo ''
@echo 'Documentation targets:'
@$(MAKE) -f $(srctree)/Documentation/DocBook/Makefile dochelp
@echo ''
@echo 'Architecture specific targets ($(ARCH)):'
@$(if $(archhelp),$(archhelp),\
echo ' No architecture specific help defined for $(ARCH)')
@echo ''
@$(if $(boards), \
$(foreach b, $(boards), \
printf " %-24s - Build for %s\\n" $(b) $(subst _defconfig,,$(b));) \
echo '')
```

```
@echo ' make V=0|1 [targets] 0 => quiet build (default), 1 => verbose
build'
@echo ' make V=2     [targets] 2 => give reason for rebuild of target'
@echo '  make O=dir [targets] Locate all output files in "dir",
including .config'
@echo ' make C=1     [targets] Check all c source with $$CHECK (sparse
by default)'
@echo ' make C=2     [targets] Force check of all c source with $$CHECK'
@echo ''
@echo 'Execute "make" or "make all" to build all targets marked with [*]
'
@echo 'For further info see the ./README file'
# 第一行定义了一个伪目标 help。
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
#


# Documentation targets
#
------------------------------------------------------------------------
------
%docs: scripts_basic FORCE
# Documentation targets
#
------------------------------------------------------------------------
------
%docs: scripts_basic FORCE
$(Q)$(MAKE) $(build)=Documentation/DocBook $@
# 定义了一个规则，目标是%docs，依赖关系是 scripts_basic FORCE
# $@是 Makefile 的自动化变量，表示当前规则的目标集。

else # KBUILD_EXTMOD

###
# External module support.
# When building external modules the kernel used as basis is considered
# read-only, and no consistency checks are made and the make
# system is not used on the basis kernel. If updates are required
# in the basis kernel ordinary make commands (without M=...) must
# be used.
#
# The following are the only valid targets when building external
# modules.
# make M=dir clean         Delete all automatically generated files
# make M=dir modules     Make all modules in specified dir
```

```
# make M=dir                Same as 'make M=dir modules'
# make M=dir modules_install
#                                          Install the modules built
in the module directory
#                                          Assumes install directory
is already created

# We are always building modules
KBUILD_MODULES := 1
PHONY += crmodverdir
crmodverdir:
$(Q)mkdir -p $(MODVERDIR)
$(Q)rm -f $(MODVERDIR)/*
# 第一行给变量 KBUILD_MODULES 赋值 1.
# 第二行给变量 PHONY 追加值。
# 第三行定义了一个伪目标 crmodverdir
# mkdir 是 linux shell 下创建文件夹的命令，其语法是：mkdir [-p] dirName
# rm 是 linux shell 下删除文件或文件夹的命令，其语法是：rm [选项]... 文
件...

PHONY += $(objtree)/Module.symvers
$(objtree)/Module.symvers:
@test -e $(objtree)/Module.symvers || ( \
echo; \
echo " WARNING: Symbol version dump $(objtree)/Module.symvers"; \
echo "                       is missing; modules will have no dependencies
and modversions."; \
echo )
# 第一行给变量 PHONY 追加值。
# 第二行定义了一个为目标$(objtree)/Module.symvers。
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
# echo 是 Makefile 中的一个将输入的字符串送往标准输出关键字，其语法是：
echo [-ne][字符串]
# test 是 linux shell 下的一个对档案侦测、判定的函数，其语法是：test [选
项] string
# 字符“\”有两个功能：一是：续行；二是：转义。

module-dirs := $(addprefix _module_,$(KBUILD_EXTMOD))
PHONY += $(module-dirs) modules
$(module-dirs): crmodverdir $(objtree)/Module.symvers
$(Q)$(MAKE) $(build)=$(patsubst _module_%,%,$@)
# 第一行将函数 addprefix 的返回值展开后赋给变量 module-dirs。
# 第二行给变量 PHONY 追加值。
# 第三行定义了一个规则，目标是$(module-dirs)，依赖文件是 crmodverdir
$(objtree)/Module.symvers
```

```
# addprefix 是 Makefile 中的一个加前缀函数函数，其语法是：$(addprefix <prefix>;,<names...>;)
# patsubst 是 Makefile 中的一个模式字符串替换函数，
# 其语法是：$(patsubst <pattern>;,<replacement>;,<text>;)
# 查找<text>;中的单词是否符合模式<pattern>;，如果匹配的话，则以<replacement>;替换。

modules: $(module-dirs)
@echo ' Building modules, stage 2.';
$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost
# 上面定义了一个规则，目标是 modules，依赖文件是$(module-dirs)。
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
# echo 是 Makefile 中的一个将输入的字符串送往标准输出关键字，其语法是：echo [-ne][字符串]

PHONY += modules_install
modules_install: _emodinst_ _emodinst_post
# 第一行给变量 PHONY 追加值
# 第二行定义了一个依赖关系，目标是 modules_install，依赖文件是 _emodinst_ _emodinst_post

install-dir := $(if $(INSTALL_MOD_DIR),$(INSTALL_MOD_DIR),extra)
PHONY += _emodinst_
_emodinst_:
$(Q)mkdir -p $(MODLIB)/$(install-dir)
$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modinst
# 第一行将 if 函数的返回值展开后赋给变量 install-dir。
# 第二行给变量 PHONY 追加值。
# 第三行定义了一个伪目标_emodinst_
# mkdir 是 linux shell 下的一个创建文件夹的命令，其语法是：mkdir [-p] dirName

# Run depmod only is we have System.map and depmod is executable
quiet_cmd_depmod = DEPMOD $(KERNELRELEASE)
cmd_depmod = if [ -r System.map -a -x $(DEPMOD) ]; then \
$(DEPMOD) -ae -F System.map                          \
$(if $(strip $(INSTALL_MOD_PATH)),                   \
-b $(INSTALL_MOD_PATH) -r)                           \
$(KERNELRELEASE);                                    \
fi
# 第一二行分别给变量 quiet_cmd_depmod 和 cmd_depmod 赋值。
# strip 是 Makefile 中的去空格函数，其语法是：$(strip <string>;)
# 字符“\”有两个功能：一是：续行；二是：转义。
```

```
PHONY += _emodinst_post
_emodinst_post: _emodinst_
$(call cmd,depmod)
# 第一行给变量 PHONY 追加值。
# 第二行定义了一个规则，目标是_emodinst_post，依赖文件是_emodinst_
# call 是 Makefile 中一个用来创建新的参数化的函数，
# 其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
# <expression>;参数中的变量，会被参数<parm1>;，<parm2>;，<parm3>;依次取代。

clean-dirs := $(addprefix _clean_,$(KBUILD_EXTMOD))
# 上面将函数 addprefix 的返回值展开后赋给变量 clean-dirs。
# addprefix 是 Makefile 中一个加前缀函数，其语法是：$(addprefix <prefix>;,<names...>;)
# 功能：把前缀<prefix>;加到<names>;中的每个单词后面。

PHONY += $(clean-dirs) clean
$(clean-dirs):
$(Q)$(MAKE) $(clean)=$(patsubst _clean_%,%,$@)
# 第一行给变量 PHONY 追加值。
# 第二行定义了一个为目标$(clean-dirs)，
# patsubst 是 Makefile 中的一个模式字符串替换函数，
# 其语法是：$(patsubst <pattern>;,<replacement>;,<text>;)
# 查找<text>;中的单词是否符合模式<pattern>;，如果匹配的话，则以<replacement>;替换。

clean: rm-dirs := $(MODVERDIR)
clean: $(clean-dirs)
$(call cmd,rmdirs)
@find $(KBUILD_EXTMOD) $(RCS_FIND_IGNORE) \
\( -name '*.[oas]' -o -name '*.ko' -o -name '.*.cmd' \
-o -name '.*.d' -o -name '.*.tmp' -o -name '*.mod.c' \) \
-type f -print | xargs rm -f
# 第一行定义了一个规则，目标是 clean，依赖文件是 rm-dirs，命令是给依赖文件 rm-dirs 赋值
# call 是 Makefile 中一个用来创建新的参数化的函数，
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
# find 是 linux shell 下的一个查找命令，其语法是：find path option [-print -exec -ok]
# 字符“\”有两个功能：一是：续行；二是：转义。

help:
@echo ' Building external modules.'
@echo ' Syntax: make -C path/to/kernel/src M=$$PWD target'
@echo ''
```

```
@echo ' modules                 - default target, build the module(s)'
@echo ' modules_install - install the module'
@echo ' clean                       - remove generated files in module
directory only'
@echo ''
# 上面定义了一个伪目标 help。
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
# echo 是 Makefile 中的一个将输入的字符串送往标准输出关键字，其语法是：
echo [-ne][字符串]

# Dummies...
PHONY += prepare scripts
prepare: ;
scripts: ;
endif # KBUILD_EXTMOD
# 第一行给变量 PHONY 追加值
# 第二三行分别定义了一个为目标，命令为空命令。

# Generate tags for editors
#
---------------------------------------------------------------------
------

#We want __srctree to totally vanish out when KBUILD_OUTPUT is not set
#(which is the most common case IMHO) to avoid unneeded clutter in the
big tags file.
#Adding $(srctree) adds about 20M on i386 to the size of the output file!

ifeq ($(src),$(obj))
__srctree =
else
__srctree = $(srctree)/
endif
#  ifeq 是 Makefile 中的一个条件关键字，其语法是：ifeq (<arg1>;, <arg2>;)
# 功能：比较参数“arg1”和“arg2”的值是否相同。

ifeq ($(ALLSOURCE_ARCHS),)
ifeq ($(ARCH),um)
ALLINCLUDE_ARCHS := $(ARCH) $(SUBARCH)
else # 这个 else 与 ifeq ($(ARCH),um)对应。
ALLINCLUDE_ARCHS := $(ARCH)
endif     # 这个 endif 与 ifeq ($(ARCH),um)对应。
else     # 这个 else 与外层的 ifeq ($(ALLSOURCE_ARCHS),)对应
#Allow user to specify only ALLSOURCE_PATHS on the command line, keeping
existing behavour.
```

```
ALLINCLUDE_ARCHS := $(ALLSOURCE_ARCHS)
endif     # 这个 endif 与外层的 ifeq ($(ALLSOURCE_ARCHS),)对应
# ifeq 是 Makefile 中的一个条件关键字，其语法是：ifeq (<arg1>;, <arg2>;)
# 功能：比较参数“arg1”和“arg2”的值是否相同。
# 上面在 ifeq ($(ALLSOURCE_ARCHS),)里面嵌入了一个 ifeq ($(ARCH),um)。

ALLSOURCE_ARCHS := $(ARCH)
# 给变量 ALLSOURCE_ARCHS 赋值

define find-sources
( for ARCH in $(ALLSOURCE_ARCHS) ; do \
find $(__srctree)arch/$${ARCH} $(RCS_FIND_IGNORE) \
-name $1 -print; \
done ; \
find $(__srctree)security/selinux/include $(RCS_FIND_IGNORE) \
-name $1 -print; \
find $(__srctree)include $(RCS_FIND_IGNORE) \
\( -name config -o -name 'asm-*' \) -prune \
-o -name $1 -print; \
for ARCH in $(ALLINCLUDE_ARCHS) ; do \
find $(__srctree)include/asm-$${ARCH} $(RCS_FIND_IGNORE) \
-name $1 -print; \
done ; \
find $(__srctree)include/asm-generic $(RCS_FIND_IGNORE) \
-name $1 -print; \
find $(__srctree) $(RCS_FIND_IGNORE) \
\( -name include -o -name arch \) -prune -o \
-name $1 -print; \
)
endef
# define 是 Makefile 中的一个定义命令包（相同的命令序列）的关键字，
# find 是 linux shell 下的一个查找命令，其语法是：find path option [-print
-exec -ok]
# 字符“\”有两个功能：一是：续行；二是：转义。

define all-sources
$(call find-sources,'*.[chS]')
endef
define all-kconfigs
$(call find-sources,'Kconfig*')
endef
define all-defconfigs
$(call find-sources,'defconfig')
endef
# define 是 Makefile 中的一个定义命令包（相同的命令序列）的关键字，
```

call 是 Makefile 中的唯一一个来创建新的参数化的函数，
其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
<expression>;参数中的变量，会被参数<parm1>;，<parm2>;，<parm3>;依次取代。

```
define xtags
if $1 --version 2>&1 | grep -iq exuberant; then \
$(all-sources) | xargs $1 -a \
-I __initdata,__exitdata,__acquires,__releases \
-I EXPORT_SYMBOL,EXPORT_SYMBOL_GPL \
--extra=+f --c-kinds=+px \
--regex-asm='/ENTRY\(([^)]*)\).*/\1/'; \
$(all-kconfigs) | xargs $1 -a \
--langdef=kconfig \
--language-force=kconfig \
--regex-kconfig='/^[[:blank:]]*config[[:blank:]]+([[:alnum:]_]+)/\1/';
\
$(all-defconfigs) | xargs -r $1 -a \
--langdef=dotconfig \
--language-force=dotconfig \
--regex-dotconfig='/^#?[[:blank:]]*(CONFIG_[[:alnum:]_]+)/\1/'; \
elif $1 --version 2>&1 | grep -iq emacs; then \
$(all-sources) | xargs $1 -a; \
$(all-kconfigs) | xargs $1 -a \
--regex='/^[ \t]*config[ \t]+\([a-zA-Z0-9_]+\)/\1/'; \
$(all-defconfigs) | xargs -r $1 -a \
--regex='/^#?[ \t]?\(CONFIG_[a-zA-Z0-9_]+\)/\1/'; \
else \
$(all-sources) | xargs $1 -a; \
fi
endef
```

define 是 Makefile 中的一个定义命令包（相同的命令序列）的关键字，
字符“\”有两个功能：一是：续行；二是：转义。
字符“|”是管道的标志，它的作用是：将上一个命令的 stdout 重定向到下一个命令的 stdin。
grep 是 linux shell 中一个在文件中查找字符串的命令。其语法是：grep 字符串 文件名
xargs 是 linux shell 中的一个对输出执行其他某些命令的命令。

```
quiet_cmd_cscope-file = FILELST cscope.files
cmd_cscope-file = (echo \-k; echo \-q; $(all-sources)) > cscope.files
```

第一二行分别给变量 quiet_cmd_cscope-file、cmd_cscope-file 赋值。
echo 是 Makefile 中的一个将输入的字符串送往标准输出关键字，其语法是：echo [-ne][字符串]

```
# 字符“\”有两个功能：一是：续行；二是：转义。
# 字符“>”是重定向标志,功能将前一个命令的 stdou 重定向到档案(文件等)。

quiet_cmd_cscope = MAKE        cscope.out
cmd_cscope = cscope -b
# 分别给变量 quiet_cmd_cscope 和 cmd_cscope 赋值。

cscope: FORCE
$(call cmd,cscope-file)
$(call cmd,cscope)
# 上面定义了一个规则，目标是 cscope，依赖文件是 FORCE。
# call 是 Makefile 中的唯一一个来创建新的参数化的函数，
# 其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)

quiet_cmd_TAGS = MAKE     $@
define cmd_TAGS
rm -f $@; \
$(call xtags,etags)
endef
# 第一行给变量 quiet_cmd_TAGS 赋值。
# $@是 Makefile 的自动化变量，它代表当前规则的目标文件集。
# define 是 Makefile 中的一个定义命令包（相同的命令序列）的关键字，
# rm 是 linux shell 下删除文件或文件夹的命令，其语法是：rm [选项]... 文
件...
# call 是 Makefile 中的唯一一个来创建新的参数化的函数，
# 其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)

TAGS: FORCE
$(call cmd,TAGS)
# 上面定义了一个规则，目标是 TAGS，依赖文件是 FORCE。
# call 是 Makefile 中的唯一一个来创建新的参数化的函数，
# 其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)

quiet_cmd_tags = MAKE     $@
define cmd_tags
rm -f $@; \
$(call xtags,ctags)
endef
# 第一行给变量 quiet_cmd_tags 赋值。
# $@是 Makefile 的自动化变量，它代表当前规则的目标文件集。
# define 是 Makefile 中的一个定义命令包（相同的命令序列）的关键字，
# rm 是 linux shell 下删除文件或文件夹的命令，其语法是：rm [选项]... 文
件...
# call 是 Makefile 中的唯一一个来创建新的参数化的函数，
# 其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)
```

```
tags: FORCE
$(call cmd,tags)
# 上面定义了一个规则，目标是 tags，依赖文件是 FORCE
# call 是 Makefile 中的唯一一个来创建新的参数化的函数，
# 其语法是：$(call <expression>;,<parm1>;,<parm2>;,<parm3>;...)


# Scripts to check various things for consistency
#
-----------------------------------------------------------------------
------
# Scripts to check various things for consistency
#
-----------------------------------------------------------------------
------

includecheck:
find * $(RCS_FIND_IGNORE) \
-name '*.[hcS]' -type f -print | sort \
| xargs $(PERL) -w scripts/checkincludes.pl
# 第一行定义了一个伪目标 includecheck
# find 是 linux shell 下的一个查找命令，其语法是：find path option [-print
-exec -ok]
# 字符“\”有两个功能：一是：续行；二是：转义。
# 字符“|”是管道的标志，它的作用是：将上一个命令的 stdout 重定向到下一
个命令的 stdin。
# xargs 是 linux shell 中的一个对输出执行其他某些命令的命令。

versioncheck:
find * $(RCS_FIND_IGNORE) \
-name '*.[hcS]' -type f -print | sort \
| xargs $(PERL) -w scripts/checkversion.pl
# 第一行定义了一个伪目标 versioncheck
# find 是 linux shell 下的一个查找命令，其语法是：find path option [-print
-exec -ok]
# 字符“\”有两个功能：一是：续行；二是：转义。
# 字符“|”是管道的标志，它的作用是：将上一个命令的 stdout 重定向到下一
个命令的 stdin。
# xargs 是 linux shell 中的一个对输出执行其他某些命令的命令。

namespacecheck:
$(PERL) $(srctree)/scripts/namespace.pl
# 上面定义了一个伪目标 namespacecheck
```

```
endif #ifeq ($(config-targets),1)
endif #ifeq ($(mixed-targets),1)
# 上面第一个 endif 与 ifeq ($(config-targets),1)对应
# 上面第二个 endif 与 ifeq ($(mixed-targets),1)对应

PHONY += checkstack kernelrelease kernelversion
# 给变量 PHONY 追加值

# UML needs a little special treatment here. It wants to use the host
# toolchain, so needs $(SUBARCH) passed to checkstack.pl. Everyone
# else wants $(ARCH), including people doing cross-builds, which means
# that $(SUBARCH) doesn't work here.
ifeq ($(ARCH), um)
CHECKSTACK_ARCH := $(SUBARCH)
else
CHECKSTACK_ARCH := $(ARCH)
endif
checkstack:
$(OBJDUMP) -d vmlinux $$(find . -name '*.ko') | \
$(PERL) $(src)/scripts/checkstack.pl $(CHECKSTACK_ARCH)
# ifeq 是 Makefile 中的一个条件关键字，其语法是：ifeq (<arg1>;, <arg2>;)
# 功能：比较参数“arg1”和“arg2”的值是否相同。
# 第二四行分别在不同的情况下给变量 CHECKSTACK_ARCH 赋值
# 第六行定义了一个伪目标 checkstack。
# 字符“\”有两个功能：一是：续行；二是：转义。
# 字符“|”是管道的标志，它的作用是：将上一个命令的 stdout 重定向到下一个命令的 stdin。

kernelrelease:
$(if $(wildcard include/config/kernel.release), $(Q)echo
$(KERNELRELEASE), \
$(error kernelrelease not valid - run 'make prepare' to update it))
kernelversion:
@echo $(KERNELVERSION)
# 第一四行分别定义了一个伪目标。
# if 是 Makefile 中函数。其语法是：$(if
<condition>;,<then-part>;,<else-part>;)
# <condition>;参数是 if 的表达式，如果其返回的为非空字符串，
# 那么这个表达式就相当于返回真，于是，<then-part>;会被计算，否则
<else-part>;会被计算。
# 当“@”字符在命令行前，那么这个命令将不被 make 显示出来。
# echo 是 Makefile 中的一个将输入的字符串送往标准输出关键字，其语法是：
echo [-ne][字符串]
```

```
# Single targets
#
------------------------------------------------------------------------
------
# Single targets are compatible with:
# - build whith mixed source and output
# - build with separate output dir 'make O=...'
# - external modules
#
# target-dir => where to store outputfile
# build-dir => directory in kernel source tree to use

ifeq ($(KBUILD_EXTMOD),)
build-dir = $(patsubst %/,%,$(dir $@))
target-dir = $(dir $@)
else
zap-slash=$(filter-out .,$(patsubst %/,%,$(dir $@)))
build-dir = $(KBUILD_EXTMOD)$(if $(zap-slash),/$(zap-slash))
target-dir = $(if $(KBUILD_EXTMOD),$(dir $<),$(dir $@))
endif
# ifeq 是 Makefile 中的一个条件关键字，其语法是：ifeq (<arg1>;, <arg2>;)
# 功能：比较参数“arg1”和“arg2”的值是否相同。
# patsubst 是 Makefile 中的模式字符串替换函数，其语法是：$(patsubst
<pattern>;,<replacement>;,<text>;)
# 功能：查找<text>;中的单词是否符合模式<pattern>;，如果匹配的话，则以
<replacement>;替换。
# filter-out 是 Makefile 中的反过滤函数，其语法是：$(filter-out
<pattern...>;,<text>;)
# 功能：以<pattern>;模式过滤<text>;字符串中的单词，去除符合模式
<pattern>;的单词。可以有多个模式。
# if 是 Makefile 中函数。其语法是：$(if
<condition>;,<then-part>;,<else-part>;)

%.s: %.c prepare scripts FORCE
$(Q)$(MAKE) $(build)=$(build-dir) $(target-dir)$(notdir $@)
%.i: %.c prepare scripts FORCE
$(Q)$(MAKE) $(build)=$(build-dir) $(target-dir)$(notdir $@)
%.o: %.c prepare scripts FORCE
$(Q)$(MAKE) $(build)=$(build-dir) $(target-dir)$(notdir $@)
%.lst: %.c prepare scripts FORCE
$(Q)$(MAKE) $(build)=$(build-dir) $(target-dir)$(notdir $@)
%.s: %.S prepare scripts FORCE
$(Q)$(MAKE) $(build)=$(build-dir) $(target-dir)$(notdir $@)
%.o: %.S prepare scripts FORCE
```

```
$(Q)$(MAKE) $(build)=$(build-dir) $(target-dir)$(notdir $@)
%.symtypes: %.c prepare scripts FORCE
$(Q)$(MAKE) $(build)=$(build-dir) $(target-dir)$(notdir $@)
# 上面定义了一系列的规则。

# Modules
/ %/: prepare scripts FORCE
$(Q)$(MAKE) KBUILD_MODULES=$(if $(CONFIG_MODULES),1) \
$(build)=$(build-dir)
%.ko: prepare scripts FORCE
$(Q)$(MAKE) KBUILD_MODULES=$(if $(CONFIG_MODULES),1)     \
$(build)=$(build-dir) $(@:.ko=.o)
$(Q)$(MAKE) -f $(srctree)/scripts/Makefile.modpost
# 上面第一四行分别定义了一个规则。
# if 是 Makefile 中函数。其语法是：$(if
<condition>;,<then-part>;,<else-part>;)

# FIXME Should go into a make.lib or something
#
================================================================================

quiet_cmd_rmdirs = $(if $(wildcard $(rm-dirs)),CLEAN     $(wildcard
$(rm-dirs)))
cmd_rmdirs = rm -rf $(rm-dirs)
# 上面两行分别给变量 quiet_cmd_rmdirs 和 cmd_rmdirs 赋值
# if 是 Makefile 中函数。其语法是：$(if
<condition>;,<then-part>;,<else-part>;)
# rm 是 linux shell 中的删除文件或文件夹的命令，语法是：rm [选项] 文件[或
文件夹]

quiet_cmd_rmfiles = $(if $(wildcard $(rm-files)),CLEAN     $(wildcard
$(rm-files)))
cmd_rmfiles = rm -f $(rm-files)
# 上面两行分别给变量 quiet_cmd_rmfiles 和 cmd_rmfiles 赋值
# if 是 Makefile 中函数。其语法是：$(if
<condition>;,<then-part>;,<else-part>;)
# rm 是 linux shell 中的删除文件或文件夹的命令，语法是：rm [选项] 文件[或
文件夹]

a_flags = -Wp,-MD,$(depfile) $(AFLAGS) $(AFLAGS_KERNEL) \
$(NOSTDINC_FLAGS) $(CPPFLAGS) \
$(modkern_aflags) $(EXTRA_AFLAGS) $(AFLAGS_$(basetarget).o)
# 第一行给变量 a_flags 赋值
# 第二三行是命令，命令都是以 tab 开头的。
```

```
quiet_cmd_as_o_S = AS            $@
cmd_as_o_S              = $(CC) $(a_flags) -c -o $@ $<
# 上面两行分别给变量 quiet_cmd_as_o_S 和 cmd_as_o_S 赋值。
# $@和$<都是 Makefile 中的自动化变量，$@表示当前规则中的目标文件集，$<表示依赖目标中的第一个目标名字。

# read all saved command lines

targets := $(wildcard $(sort $(targets)))
cmd_files := $(wildcard .*.cmd $(foreach f,$(targets),$(dir $(f)).$(notdir $(f)).cmd))
# 上面两行分别给变量 targets 和 cmd_files 赋值
# sort 是 Makefile 中的一个排序（升序）函数，其语法是：$(sort <list>;)
# wildcard 是 Makefile 中的关键字，它的作用是让通配符在变量中展开。
# foreach 是 Makefile 中的循环函数，其语法是：$(foreach <var>;,<list>;,<text>;)
# dir 是 Makefile 中的取目录函数，其语法是：$(dir <names...>;)
# 功能：从文件名序列<names>;中取出目录部分，即最后一个反斜杠（“/”）之前的部分。若没有反斜杠则返回“./”。

ifneq ($(cmd_files),)
$(cmd_files): ; # Do not try to update included dependency files
include $(cmd_files)
endif
# ifeq 是 Makefile 中的一个条件关键字，其语法是：ifeq (<arg1>;, <arg2>;)
# 功能：比较参数“arg1”和“arg2”的值是否相同。
# include 是 Makefile 中的关键字，其语法是：include <filename>;
# 把别的 Makefile 包含进来，这很像 C 语言的#include，被包含的文件会原模原样的放在当前文件的包含位置。

# Shorthand for $(Q)$(MAKE) -f scripts/Makefile.clean obj=dir
# Usage:
# $(Q)$(MAKE) $(clean)=dir
clean := -f $(if $(KBUILD_SRC),$(srctree)/)scripts/Makefile.clean obj
# 上面这一行是给变量 clean 赋值
# if 是 Makefile 中函数。其语法是：$(if <condition>;,<then-part>;,<else-part>;)

endif # skip-makefile

PHONY += FORCE
FORCE:
# 第一行给变量 PHONY 追加值
# 第二行定义了一个伪目标 FORCE
```

```
# Cancel implicit rules on top Makefile, `-rR' will apply to sub-makes.
Makefile: ;
# 上面这一行定义了一个伪目标 Makefile，命令为空

# Declare the contents of the .PHONY variable as phony. We keep that
# information in a variable se we can use it in if_changed and friends.
.PHONY: $(PHONY)
# 上面这一行定义了一个依赖关系。
```