西北狼

stone.blog.chinaunix.net

重要的不是答案，重要的是思考过程！

小卒-IT

博客访问： 31988
博文数量： 28
博客积分： 632
博客等级： 上士
技术积分： 231
用 户 组： 普通用户
注册时间： 2010-12-19 19:40

Makefile 语法分析

2011-04-13 09:59:21

分类： LINUX

Makefile 语法分析 第一部分

```
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mak
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 = O，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.
```


上一篇：Linux 2.6内核Makefile分析
下一篇：一个高手毕生精力总结的电脑技巧

OpenWRT之自建IPK MakeFile规...

openwrt: Makefile 框架分析...

如何利用percona-toolkit工具...

httpd...

xmanager 2.0 for linux配置

什么是shell

linux socket的bug??

shell怎么读取网页内容...

ssh等待连接的超时问题...

curl: (56) Recv failure: Con...

给主人留下些什么吧！~~

评论热议

请登录后评论。

登录 注册