NAME

makerules – system programmers guide for compiling projects on different platforms

SYNOPSIS

SRCROOT= ..
RULESDIR= RULES
include $(SRCROOT)/$(RULESDIR)/rules.top
local defines are here
include $(SRCROOT)/$(RULESDIR)/rules.*

See chapter CURRENTLY SUPPORTED TARGET TYPES for possible values of **rules.***.

DESCRIPTION

Makerules is a set of rules that allows compiling of structured projects with small and uniformly structured makefiles. All rules are located in a central directory. Compiling the projects on different platforms can be done without the need to modify any of the makefiles that are located in the projects directories.

Three make programs are currently supported: *Sunpro make*, *GNU make* and *smake*. If you want to add support for other make programs, read the sections about the minimum requirements for a make program and about the structure of the **make rule** system.

This manual will help programmers who need to make modifications on the make rule system itself. If you want to know something on how to use the **makefile system** have a look at **makefiles**(4).

The main design goal was to have no definition on more than place in the make rules. This implies that system programmers who want to add or modify rules must follow this goal in order not to destroy functionality in other places.

The visible result for the user is a set of small and easy to read makefiles, each located in the project's leaf directory and therefore called *leaf*-makefile.

Each of these *leaf*-makefiles, in fact contains no rule at all. It simply defines some macros for the *make*-program and includes two files from a central make rule depository. These included files define the rules that are needed to compile the project.

Each *leaf*-makefile is formed in a really simple way:

- It first defines two macros that define the relative location of the project's root directory and the name of the directory that contains the complete set of of rules and then includes the rule file *rules.top* from the directory that forms the central rule depository. You only have to edit the macro *SRCROOT* to reflect the relative location of the project's root directory.
- The next part of a *leaf*-makefile defines macros that describe the target and the source. You can only have one target per *leaf*-makefile. Of course, there may be many source files, that are needed to create that target. If you want to make more than one target in a specific directory, you have to put more than one makefile into that directory. This is the part of a makefile that describes a unique target. Edit this part to contain all source files, all local include files and all non global compile time flags that are needed for your target. For a typical target this is as simple as filling in a form.
- Each *leaf*-makefile finally includes a file from the rules directory that contains rules for the appropriate type of target that is to be made from this *leaf*-makefile.

The makefile in each directory has to be called *Makefile*. If you want to have more than one makefile in a specific directory, you have to choose different names for the other makefiles.

Currently Supported Target Types

There are rules for the following type of targets:

commands
: The make rules for user level commands like *cat*, *ls* etc. are located in the file *rules.cmd*

drivers	The make rules for device drivers are located in the file *rules.drv*
libraries	The make rules for non shared libraries are located in the file *rules.lib*
shared libraries	The make rules for shared libraries are located in the file *rules.shl*
localized files	The make rules for localized files are located in the file *rules.loc*
nonlocalized files	The make rules for non localized files are located in the file *rules.aux*
shell scripts	The make rules for shell scripts (a variant of localized files) are located in the file *rules.scr*
manual pages	The make rules for manual pages (a variant of localized files) are located in the file *rules.man*
diverted makefiles	The make rules for projects that need to have more than one makefile in a specific directory are located in the file *rules.mks* It contains a rule that diverts to the listed sub makefiles. Each sub makefile may be of any type.
directories	The make rules for sub directories are located in the file *rules.dir*

Minimum Requirements For A Make Program

The make rules currently have support for *Sunpro make*, *GNU make* and *smake*. If you like to add support for other make programs, they need to have some minimal features that go beyond the capabilities of the standard UNIX **make** program. *BSDmake* could be supported if it supports pattern matching rules correctly.

include
: The make program must be able to recursively include other files from within a *makefile.* The name if the file to include must be allowed to be a macro. The make program must be able to do this in a way that if the file that should be included may be a result of make rule. e.g if the file to be included does not exist or is outdated, it should be built before an attempt is made to actually include it.

appending to a macro
: A macro reference of the form:

macro += addval

should append **addval** to the string that is currently in **macro**.

suffix macro replacement
: A macro reference of the form:

out= $(macro : string1 = string2)

should replace a suffix *string1* to *string2* in all words that are in **macro**, where string1 is either a suffix, or a word to be replaced in the macro definition, and string2 is the replacement suffix or word. *String1* and *string2* must be replaced correctly even if they are macros themselves. Words in a macro value are separated by SPACE, TAB, and escaped NEWLINE characters.

pattern macro replacement
: A macro reference of the form:

out= $(macro : op%os = np%ns)

should replace a central pattern in **macro**, where **op** is the existing (old) prefix and **os** is the existing (old) suffix, **np** and **ns** are the new prefix and new suffix, respectively, and the pattern matched by % (a string of zero or more characters), is carried forward from the value being replaced. For example:

PROGRAM=fabricate

DEBUG= $(PROGRAM:%=tmp/%−g)

sets the value of DEBUG to tmp/fabricate−g. *Op*, *os*, *np* and *ns* must be replaced correctly even if they are macros themselves.

Understanding Basic Algorithms

One of the basic algorithms used in the make rule system is needed to set an undefined macro to a guaranteed default value. Because not all make programs have support for *if then else* structures, a different method has to be used.

The method used in **make rules** is implemented by using **suffix macro replacement** and **pattern macro replacement**.

First, a macro that contains a unique suffix is defined:

```
# Define magic unique cookie
_UNIQ=                  .XxZzy−
```

This macro is used for all places where it is necessary to have a macro with a guaranteed default value. The following example shows the basic algorithm that is used to implement the phrase: **If** *$(MAKE_NAME)* contains a value, **then** *$(XMAKEPROG)* will be set to *$(MAKE_NAME)* **else** *$(XMAKEPROG)* will be set to *$(MAKEPROG)*.

```
_MAKEPROG=          $(_UNIQ)$(MAKE_NAME)
__MAKEPROG=         $(_MAKEPROG:$(_UNIQ)=$(MAKEPROG))
XMAKEPROG=          $(__MAKEPROG:$(_UNIQ)%=%)
```

The first line in this example, sets the macro *_MAKEPROG* to the concatenation of the value of *MAKE_NAME* and **.XxZzy−**. If the macro *MAKE_NAME* is empty at this time, *_MAKEPROG* will contain only **.XxZzy−**.

In the second line, *__MAKEPROG* is set to the value of *_MAKEPROG*. If *_MAKEPROG* contains only **.XxZzy−** this implies, that **.XxZzy−** is the suffix. This suffix is then replaced by the value of *MAKEPROG*, in this case *__MAKEPROG* will contain the unmodified value of *MAKEPROG*. If *_MAKEPROG* contains a concatenation of **.XxZzy−** and something else, **.XxZzy−** will not be a suffix, but a prefix of *_MAKEPROG* and for this reason *__MAKEPROG* will contain the unmodified value of *_MAKEPROG*, which is a concatenation of **.XxZzy−** and the value of *MAKE_NAME*.

In the third line, *XMAKEPROG* is set to the value of *__MAKEPROG*. If *__MAKEPROG* has the prefix **.XxZzy−** at this time, **.XxZzy−** is stripped of.

The Structure in Make Macro names

The names used for **make macros** are structured in a way that allows to use **grep**(1) to look for the names in the **make rules.** To allow this, no name must be a substring of another name.

If a command needs options that have to be specified in macros, there is a **make macro** that is named *XXXFLAGS.* This is compliant to usual make file rules. The are internal **make macros** called *XXXOPTS* and *XXXOPTX* that will be combined for *XXXFLAGS*:

LDFLAGS= $(LDOPTS) $(LDOPTX)

Where *XXXOPTS* is the name of the macro that is used internally and *XXXOPTX* is the name of the macro that may be used from the command line of the make program. *XXXOPTX* therefore is used to append to the content of *XXXFLAGS* If the value of *XXXFLAGS* need to be overwritten, *XXXOPTS* may be used within the command line flags of the make program.

The Structure Of The Make Rule System

The Structure Of The Basic Rules in rules.top

The file **RULES/rules.top** first includes a rule file that depends on the make program that is used. The name of this file is **RULES/mk−***makeprog***.id** where *makeprog* has to be replaced by the real name of the makeprogram e.g. **make**, **gmake**, **smake**. The purpose of this file is to set up a list of macros that identify the system where the project is currently built. These macros have values that contain only lower case letters and define:

the processor architecture
: If two systems run the same operating system, this is a unique value if a simple user level program will not need to be recompiled in order to run on the other system. Possible values are **sparc**, **mc68020**, **pentium**. This is the output of **uname −p**. The value is stored in **P_ARCH**.

the kernel architecture
: If two systems may use the same value for **P_ARCH** but a heavily system dependent user level program need to be recompiled in order to run on the other system, These two systems have different kernel architectures. This is the output of **uname −m**. Possible values are **sun3**, **sun4c**, **sun4m**. The value is stored in **K_ARCH**.

the machine architecture
: An outdated macro that is useful only on sun systems. Do not use this, use **P_ARCH** instead. This is the output of **arch**. Possible values are **sun3**, **sun4**. The value is stored in **M_ARCH**.

the hostname
: The name of the machine where the compilation takes place. This is the output of **uname −n**. The value is stored in **HOSTNAME**.

the name of the operating system
: This is the output of **uname −s**. Possible values are **sunos**, **dgux**, **hp−ux**, **irix**. The value is stored in **OSNAME**.

the release of the operating system
: This is the output of **uname −r**. Possible values are **5.5**, **4.1.4**. The value is stored in **OSREL**.

The next file to be included from **RULES/rules.top** is **RULES/os−***operating system***.id**. It defines the macros **O_ARCH** and **−O_ARCH** and may modify one of the macros that are defined in **RULES/mk−***makeprog***.id**. The macros **O_ARCH** and **−O_ARCH** are used to distinguish between different operating systems. The names of the compiler configuration files have **−O_ARCH** as a central part. On some operating systems e.g. **SunOS** and **DG−UX** it is necessary to distinguish between **SunOS 4.x** and **SunOS 5.x** or **DG−UX 3.x** and **DG−UX 4.x.**

The next file to be included from **RULES/rules.top** is **Defaults**. It defines the macros **DEFCCOM** , **DEFINCDIRS** , **LDPATH** , **RUNPATH** , **INS_BASE** and **INS_KBASE**. If the definitions have to be different on different systems, this file may contain a line int the form:

include *$(SRCROOT)***/Defaults.***$(O_ARCH)*

The actual definitions then have to be moved into these files.

Next, after setting up some internal defaults, **RULES/rules.top** includes the compiler configuration file with the name:

$(SRCROOT)/$(RULESDIR)/$(XARCH).rul

This file contains all necessary **system dependent** stuff that is needed to configure the C-compiler on the appropriate system. It is a bad idea to create a new one from scratch. Have a look at the other compiler configuration files and modify a similar file for your needs. Note that there are basically two criterias to that are important in a compiler configuration file. One is whether the system uses the *ELF* header format or not. The other is whether the system uses *shared libraries* or not.

The Structure Of The Application Specific Rules

The application specific rule files are designed in such a way that they include all necessary stuff that is needed for that specific task. The application specific rule files are:

$(RULES)/rules.aux	Rules for installing non localized auxiliary files.
$(RULES)/rules.cmd	Rules for commands like *sh.*
$(RULES)/rules.dir	Rules for sub directories.
$(RULES)/rules.drv	Rules for lodable drivers.
$(RULES)/rules.lib	Rules for static libraries.
$(RULES)/rules.loc	Rules for installing localized auxiliary files.
$(RULES)/rules.man	Rules for installing localized manual pages.
$(RULES)/rules.mks	Rules for sub makefiles.
$(RULES)/rules.mod	Rules for lodable stream modules.
$(RULES)/rules.scr	Rules for installing localized shell scripts.
$(RULES)/rules.shl	Rules for shared libraries.

Understanding The Structure Of The Make Rule System

To understand the structure of the **make rule** system while doing changes, try to use the **−xM** flag in the **smake** program. This flag will print out the include dependency list (i.e. a list that tell you which make rules is included from which other rule).

Note that some of the rules are make program dependent. If you want to make changes to these rules you may need to place the definitions into separate rule files each for the appropriate make program. Have a look into the **RULES** directory for some examples.

FILES

. . ./RULES/*
. . ./DEFAULTS/*
. . ./TARGETS/*
. . ./TEMPLATES/*

SEE ALSO

makefiles(4), **make**(1), **gmake**(1), **smake**(1).

DIAGNOSTICS

Diagnostic messages depend on the make program. Have a look at the appropriate man page.

NOTES

The make rules can be used with *Sunpro make*, *Gnu make* and *smake*. Although Gnu make runs on many platforms, it has no useful debug output.

Use *Sunpro make* or *smake* if you have problems with a makefile. *Sunpro make* and *smake*, both have a −D flag, that allows you to watch the makefiles after the first expansion. Use this option, if you are in doubt if your makefile gets expanded the right way and if the right rules are included. There is also a −d option that gives debugging output while make is running. If you want more output, use −dd, −ddd and so on.

Smake has an option −xM that shows you the include dependency for make rules.

BUGS

Source Tree Hierarchy

The following outline gives a quick tour through a typical source hierarchy:

.../ root directory of the source tree

Makefile
the top Makefile

Defaults
default definitions for that source tree. System dependent definitions are in **.../DEFAULTS/**

Targetdirs
a file containing a list of directories that are needed for that project. If the system needs different target lists depending on the target system architecture , use target specific files in **.../TARGETS/**

. . .

.../RULES/
the location of makefiles (included rules)

rules.top
the mandatory include rules (needed to setup basic rules)

rules.aux
rules needed to install a non localized auxiliary file

rules.cmd
rules needed to make an ordinary command (like /bin/sh)

rules.drv
rules needed to make a device driver

rules.lib
rules needed to make a standard (nonshared) library

rules.loc
rules needed to install a localized auxiliary file

rules.man
rules needed to install a localized manual page

rules.scr
rules needed to install a localized shell script

rules.shl
rules needed to make a shared library

rules.mks
rules needed to make more than one target in a specific directory

rules.dir
rules needed to make targets that are located in sub directories to the current directory

. . .

.../DEFAULTS/
default definitions for various target architectures are located in this directory. Templates for some architectures can be found in the *.../TEMPLATES/* directory.

.../TARGETS/
target list definitions for various target architectures are located in this directory.

.../TEMPLATES/
templates that should be used inside the project (rename to Makefile, if it is the only makefile on that directory, rename to *target.mk,* if there is more than one target in that directory)

Defaults
Defaults file for the source root directory

Defaults.linux
Defaults file for *linux*. This sould be installed in the **.../DEFAULTS/** directory.

Makefile.root
Makefile for the source root directory

Makefile.aux
: Makefile for a non localized auxiliary file

Makefile.cmd
: Makefile for an ordinary command (like /bin/sh)

Makefile.lib
: Makefile for a standard (nonshared) library

Makefile.loc
: Makefile for a localized auxiliary file

Makefile.man
: Makefile for a localized manual page

Makefile_de.man
: Makefile for a localized manual page in the german locale

Makefile.scr
: Makefile for a localized shell script

Makefile.shl
: Makefile for a shared library

Makefile.drv
: Makefile for a device driver

Makefile.mks
: Makefile for more than one target in a specific directory

Makefile.dir
: Makefile for targets that are located in sub directories to the current directory

. . .

.../cmd/
source tree for normal commands

Makefile
the makefile for the *cmd* sub directory

Targetdirs.sun4m
a file containing a list of directories like *myprog* (see below) that are needed for that specific architecture.

myprog/
directory where the sources for a specific command are located

Makefile
makefile for *myprog*

Makefile.man
makefile for the manual page of *myprog*

mprog.c
source for myprog

mprog.tr
troff source for the manual page of myprog

OBJ/ directory where system specific sub directories are located

sparc–sunos5–cc/
directory for binaries that belong to a specific system

. . .

. . .

. . .

.../lib/ directory where the sources for a libraries are located

Makefile
the makefile for the *lib* sub directory

Targetdirs.sun4m
a file containing a list of directories like *libfoo* (see below) that are needed for that specific architecture.

libfoo/ directory where all source files for libfoo are located

. . .

.../kernel

directory for kernel modules

Makefile

the makefile for the *kernel* sub directory

Targetdirs.sun4m

a file containing a list of directories like *drv* (see below) that are needed for that specific architecture.

drv/ directory where drivers are located

Makefile

the makefile for the *drv* sub directory

Targetdirs.sun4m

a file containing a list of directories like *mydrv* (see below) that are needed for that specific architecture.

mydrv/

source for a specific driver

. . .

. . .

.../include

directory for global include files that are used in that project

.../bins directory for binary programs that are created/needed while compiling the project

sparc−sunos5−cc/

directory for binaries that belong to a specific system

. . .

.../libs directory for libraries that are created/needed while compiling the project

sparc−sunos5−cc/

directory for libraries that belong to a specific system

. . .

.../incs directory for include files that are created/needed while compiling the project

sparc−sunos5−cc/

directory for include files that belong to a specific system

. . .

. . .

AUTHOR

Jörg Schilling
Seestr. 110
D-13353 Berlin
Germany

Mail bugs and suggestions to:

joerg@schily.isdn.cs.tu-berlin.de or **js@cs.tu-berlin.de** or **jes@fokus.gmd.de**