

GRUB2 Modules

Finnbarr P. Murphy

(fpm@fpmurphy.com)

GRUB2 has a modular architecture with support for loading and unloading code modules as needed. Modules are similar to Linux kernel modules in some respects but in other ways they are like plugins for Mozilla Firefox or the [Eclipse](#) IDE. In this post I provide an overview of what a GRUB2 module is and show you how to modify an existing module to add new functionality. I also demonstrate how to develop, integrate and test your own custom module using the GRUB2 *grub-emu* userspace emulator.

The current version of GRUB2 (v1.98) has 159 modules:

acpi.mod	date.mod	gcry_shal.mod	loopback.mod	pb
kdf2.mod	terminfo.mod	gcry_sha256.mod	lsmmmap.mod	pc
affs.mod	datetime.mod	gcry_sha512.mod	ls.mod	pl
i.mod	test.mod	gcry_tiger.mod	lspci.mod	pn
afs_be.mod	dm_nv.mod	gcry_twofish.mod	lvm.mod	pr
ay.mod	tga.mod	gcry_whirlpool.mod	mdraid.mod	px
afs.mod	drivemap.mod	gettext.mod	memdisk.mod	px
g.mod	trig.mod	gfxmenu.mod	memrw.mod	ra
aout.mod	echo.mod	gfxterm.mod	minicmd.mod	ra
obe.mod	true.mod	gptsync.mod	minix.mod	ra
ata.mod	efiemu.mod	gzio.mod	mmap.mod	re
ecmd.mod	udf.mod	halt.mod	msdospart.mod	re
ata_pthru.mod	elf.mod	handler.mod	multiboot2.mod	re
e.mod	ufs1.mod	hashsum.mod	multiboot.mod	re
at_keyboard.mod	example_functional_test.mod	vbeinfo.mod	normal.mod	sc
id5rec.mod	ufs2.mod	gcry_arcfour.mod	ntfscomp.mod	se
befs_be.mod	ext2.mod	vbetest.mod	help.mod	se
id6rec.mod	uhci.mod	gcry_cast5.mod	hexdump.mod	se
befs.mod	extcmd.mod	gcry_crc.mod	hfs.mod	se
id.mod	usb_keyboard.mod	gcry_des.mod	hfsplus.mod	se
biosdisk.mod	fat.mod	video_fb.mod	iso9660.mod	se
ad.mod	usb.mod	gcry_md4.mod	jpeg.mod	se
bitmap.mod	font.mod	videotest.mod	jfs.mod	se
boot.mod	usbms.mod	gcry_md5.mod	xfs.mod	se
bitmap_scale.mod	fshelp.mod	vbefrom.mod	gcry_rfc2268.mod	sf
iserfs.mod	usbtest.mod	functional_test.mod	xnu.mod	
blocklist.mod	functional_test.mod	vbeinfo.mod		
locator.mod	vbeinfo.mod	gcry_md4.mod		
boot.mod	gcry_md5.mod	videotest.mod		
si.mod	vbefrom.mod	gcry_md5.mod		
bsd.mod	functional_test.mod	vbefrom.mod		
arch_fs_file.mod	vbetest.mod	gcry_md5.mod		
bufio.mod	gcry_camellia.mod	vbefrom.mod		
arch_fs_uuid.mod	vga.mod	functional_test.mod		
cat.mod	gcry_cast5.mod	vbeinfo.mod		
arch_label.mod	vga_text.mod	gcry_md4.mod		
chain.mod	gcry_crc.mod	videotest.mod		
arch.mod	video_fb.mod	gcry_md5.mod		
charset.mod	gcry_des.mod	vbefrom.mod		
rial.mod	video.mod	functional_test.mod		
cmp.mod	gcry_md4.mod	vbeinfo.mod		
tjmp.mod	videotest.mod	gcry_md5.mod		
configfile.mod	gcry_md5.mod	videotest.mod		
tpci.mod	xfs.mod	vbefrom.mod		
cpio.mod	gcry_rfc2268.mod	functional_test.mod		
s.mod	xnu.mod	vbeinfo.mod		

GRUB2 Modules

cpuid.mod	gcry_rijndael.mod	keystatus.mod	part_sun.mod	sh.
mod	xnu_uuid.mod			
crc.mod	gcry_rmd160.mod	linux16.mod	parttool.mod	sl
eep.mod		linux.mod	password.mod	ta
crypto.mod	gcry_seed.mod		loadenv.mod	password_pbkdf2.mod te
r.mod				
datehook.mod	gcry_serpent.mod			
rminal.mod				

Some of the more noteworthy modules are:

- *ls, cat, echo, cmp*: Provide similar functionality to their Unix command counterparts.
- *ext2, iso9660, reiserfs, xfs*: Provide support for filesystems of the same name.
- *part_sun, part_gpt, part_msdos*: Provide support for various partition schemes.
- *linux*: Loader for Linux images
- *vga, tga, vbe, png,jpeg*: Provide support for graphics and background images.

The commands to load (*insmod*) or unload (*rmmmod*) a module into GRUB2 have the same names as in GNU/Linux.

When a module is loaded, or a linked-in module is initialized, the module registers one or more commands, and can also register variables, parsers, and drivers. A GRUB2 command such as *hexdump* requires a mapping from a name to a specific module function. The file */boot/grub/command.lst* contains the mappings from the command name to the module that contains the function (and code to implement that command). Here is part of the *command.lst* file:

```
cat: minicmd
chainloader: chain
clear: minicmd
cmp: cmp
.: configfile
configfile: configfile
*cpuid: cpuid
crc: crc
date: date
*drivemap: drivemap
dump: minicmd
*echo: echo
hexdump: hexdump
```

In many cases the command name is the same name as the module name, e.g. *hexdump* and *hexdump.mod*. In other cases a loaded module may register multiple commands, e.g. *loadenv.mod* registers the *load_env* and *save_env*. If a command is entered (either directly or via a script) the appropriate module(s) is loaded if the command is not already registered. A variable is a mapping from a name to a variable defined in the module. Scripts can access the variable as \$name. Module variable eventing is supported; a module can get a callback when a value is assigned to a variable.

Modules are actually ELF format files. The main GRUB2 file *core.img* contains the necessary code to parse the ELF headers, load the module and do dynamic-linking (i.e., resolve calls to functions and variables exported by the *core.img* for use by modules). A module can be a built-in, i.e. part of *core.img* or dynamically loaded using *insmod*, or it can be loaded automatically when another module is loaded via *insmod* because the module being loaded has a dependency on it.

Dependency mappings are listed in the file *moddep.lst*. Here is part of this file:

```
video:
```

```

font: bufio video
hfsplus: fshelp
gettext: normal gzio
extcmd:
normal: crypto boot terminal charset
hashsum: extcmd normal crypto
search: extcmd search_label search_fs_uuid search_fs_file
xnu_uuid: gcry_md5
drivemap: extcmd boot mmap
password: crypto normal
read:

```

From the above you can see that the *password* module has dependencies on the *crypto* and *normal* modules. Note that the *normal* module is automatically loaded by *core.img* at startup and thus the modules *crypto*, *boot*, *terminal* and *charset* are also loaded at that time because *normal* has dependencies on these modules.

The source code for GRUB2 is fairly well structured and is mostly ANSI C code. Naturally there is a certain amount of assembler code (that is the nature of the beast) but the amount is small and you should never have any need to modify or even look at such code. It runs in 32-bit real mode and is single-threaded. In general there are no synchronization issues or races conditions to consider.

Turning now to the question of how to create and build a GRUB2 module. The obligatory *Hello World* module (*./hello/hello.mod*) is included with the GRUB2 source code tarball. Here is the source code for this module:

```

/* hello.c - test module for dynamic loading */
/*
 * GRUB -- GRand Unified Bootloader
 * Copyright (C) 2003,2007 Free Software Foundation, Inc.
 * Copyright (C) 2003 NIIIBE Yutaka <gniibe@m17n.org>
 *
 * GRUB is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * GRUB is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with GRUB. If not, see <http://www.gnu.org/licenses/>.
 */
#include <grub/types.h>
#include <grub/misc.h>
#include <grub/mm.h>
#include <grub/err.h>
#include <grub/dl.h>
#include <grub/extcmd.h>
#include <grub/i18n.h>
static grub_err_t
grub_cmd_hello (struct grub_extcmd *cmd __attribute__ ((unused)),
                int argc __attribute__ ((unused)),
                char **args __attribute__ ((unused)))
{
    grub_printf ("Hello World\n");
    return 0;
}
static grub_extcmd_t cmd;
GRUB_MOD_INIT(hello)

```

```
{
    cmd = grub_register_extcmd ("hello", grub_cmd_hello, GRUB_COMMAND_FLAG_BOTH,
                                0, N_("Say \"Hello World\"."), 0);
}
GRUB_MOD_INIT(hello)
{
    grub_unregister_extcmd (cmd);
}
```

The above C language code should not be difficult to understand. The definitions for *GRUB_MOD_INIT* and *GRUB_MOD_FINI* are in the header *dl.h*. These functions are basically used to register and unregister a command, in this case *hello*. When you enter the command *hello* the function *grub_cmd_hello* in the dynamically loaded module *hello.mod* is invoked. Never try to use *libc* library functions in a module. All the functions you should need are available in the source tarball and have the same name as the equivalent *libc* function except that they are prefixed by *grub_*, e.g. *grub_printf()* and *printf()*, *grub_strcpy()* and *strcpy*. In addition do not include regular headers such as *stdio.h*.

If you are not going to pass any arguments into a command, you must specifically declare this fact in the function declaration as follows:

```
function_name (struct grub_extcmd *cmd __attribute__ ((unused)),
               int argc __attribute__ ((unused)),
               char **args __attribute__ ((unused)))
```

The necessary build directives are already present for the *hello* module. After compiling and installing GRUB2, you should find that the *hello* module was installed in */boot/grub*. At the GRUB2 command prompt, you can load and execute the *hello* command as follows:

```
grub> insmod hello.mod
grub> hello
Hello World
grub>
```

Turning now to the *hexdump* command. The source code for this command is in the *./commands* subdirectory. Currently there is no option to dump the entire contents of a file in this command so I have decided to add this functionality. I see from the following code snippet that currently *hexdump* can take two optional arguments:

```
static const struct grub_arg_option options[] = {
    {"skip", 's', 0, N_("Skip offset bytes from the beginning of file."), 0, ARG_TYPE_INT},
    {"length", 'n', 0, N_("Read only LENGTH bytes."), 0, ARG_TYPE_INT},
    {0, 0, 0, 0, 0, 0}
};

static grub_err_t
grub_cmd_hexdump (grub_extcmd_t cmd, int argc, char **args)
{
    struct grub_arg_list *state = cmd->state;
    grub_size_t size, length;
    grub_disk_addr_t skip;
    ...
    skip = (state[0].set) ? grub strtoull (state[0].arg, 0, 0) : 0;
    length = (state[1].set) ? grub strtoul (state[1].arg, 0, 0) : 256;
```

Note that `state[0]` refers to the first option argument defined in `options` array, i.e. `skip`, and `state[1]` refers to the second option argument, i.e. `length`. Thus if something like `-s 200` is passed as a command line argument to the `hexdump` command, `state[0]` is set to a non-zero value. I use this fact to decide when to hexdump the complete file. This I do if both `state[0]` and `state[1]` are set to 0.

Here is the source code for the new functionality:

```
.....
if (!grub_strcmp (args[0], "(mem)"))
    hexdump (skip, (char *) (grub_addr_t) skip, length);
/* new functionality */
else if (state[0].set == 0 && state[1].set == 0)
{
    file = grub_gzfile_open (args[0], 1);
    if (! file)
        return grub_errno;
    skip = 0;
    while ((size = grub_file_read (file, buf, 256)) > 0
        && key != GRUB_TERM_ESC)
    {
        hexdump (skip, buf, size);
        skip += size;
        while (grub_checkkey () >= 0 &&
            (key = GRUB_TERM_ASCII_CHAR (grub_getkey ())) != GRUB_TERM_ESC)
            ;
    }
    grub_putchar ('\n');
    grub_refresh ();
    grub_file_close (file);
}
/* end new functionality */
else if ((args[0][0] == '(') && (args[0][namerlen - 1] == ')'))
....
```

The code should be pretty much self explanatory. Note that `hexdump` is a separate library function which formats and displays all or part of whatever is the `buf` buffer (depends on the numeric values in `skip` and `size`). Also, the inner `while` loop is activated if the `pager` environmental variable is set. In this case, `hexdump` displays a ---MORE--- prompt and only displays the next screen of data when a key is pressed. No changes to the build system are required; you can just rebuild GRUB2, copy `hexdump.mod` to `/boot/grub`, reboot your system and test `hexdump` from the GRUB2 command prompt.

Next we develop a brand new module called `colortest` from scratch. This module will display all the color combinations supported by GRUB2 (it the same set as for GRUB Legacy) in order to check that they are correctly displayed by GRUB2.

Here is the source code for the module. Note for the sake of brevity I have left out most of the possible color combinations as these would only add several hundred additional lines of repetitive code to the listing. The source code is commented where I think an explanation is warranted.

```
#include <grub/env.h>
#include <grub/types.h>
#include <grub/dl.h>
#include <grub/misc.h>
#include <grub/mm.h>
#include <grub/font.h>
#include <grub/term.h>
#include <grub/command.h>
```

```

#include <grub/extcmd.h>
#include <grub/i18n.h>
/* takes one optional argument -c */
static const struct grub_arg_option options[] =
{
    {"clear", 'c', 0, N_("Clear the screen first."), 0, 0},
    {0, 0, 0, 0, 0, 0}
};

/* since argc and argv are not used within the function they */
/* must be declared unused - otherwise the build fails */
static grub_err_t
grub_cmd_colortest (grub_extcmd_t cmd,
                     int argc __attribute__ ((unused)),
                     char **args __attribute__ ((unused)))
{
    struct grub_arg_list *state = cmd->state;
    grub_err_t err = GRUB_ERR_NONE;
    char color_normal[50];
    const char *tmp;
    /* save the current color string for color_normal. It is in the environment */
    /* set default if none in the environment otherwise error messages in grub-emu */.
    tmp = grub_env_get("color_normal");
    if (*tmp)
        grub_strncpy(color_normal, tmp);
    else
        grub_strncpy(color_normal, "light-gray/black");
    /* clear the screen if -c passed on command line */
    if (state[0].set)
        grub_cls();
    /* set the current color state to normal just in case it is currently highlight */
    grub_setcolorstate (GRUB_TERM_COLOR_NORMAL);
    /* black */
    /* change the color_normal setting in the environment */
    grub_env_set ("color_normal", "dark-gray/black");
    /* print this string in dark-gray on black. Black as background is transparent */
    grub_printf (" dark-gray/black      ");
    grub_env_set ("color_normal", "light-blue/black");
    grub_printf (" light-blue/black      ");
    grub_env_set ("color_normal", "light-green/black");
    grub_printf (" light-green/black      ");
    grub_env_set ("color_normal", "light-cyan/black");
    grub_printf (" light-cyan/black      \n");
    grub_env_set ("color_normal", "light-red/black");
    grub_printf (" light-red/black      ");
    grub_env_set ("color_normal", "light-magenta/black");
    grub_printf (" light-magenta/black      ");
    grub_env_set ("color_normal", "yellow/black");
    grub_printf (" yellow/black      ");
    grub_env_set ("color_normal", "white/black");
    grub_printf (" white/black      \n");
    /* blue */
    grub_env_set ("color_normal", "dark-gray/blue");
    grub_printf (" dark-gray/blue      ");
    grub_env_set ("color_normal", "light-blue/blue");
    grub_printf (" light-blue/blue      ");
    grub_env_set ("color_normal", "light-green/blue");
    grub_printf (" light-green/blue      ");
    grub_env_set ("color_normal", "light-cyan/blue");
    grub_printf (" light-cyan/blue      \n");
    grub_env_set ("color_normal", "light-red/blue");
    grub_printf (" light-red/blue      ");
    grub_env_set ("color_normal", "light-magenta/blue");
    grub_printf (" light-magenta/blue      ");
    grub_env_set ("color_normal", "yellow/blue");
    grub_printf (" yellow/blue      ");
    grub_env_set ("color_normal", "white/blue");
    grub_printf (" white/blue      \n");
    /* revert to the original color_normal */

```

```

grub_env_set ("color_normal", color_normal);
grub_printf ("\n\n");
/* black highlight */
/* change the color_highlight setting in the environment */
grub_env_set ("color_highlight", "black/blue");
/* enable color_highlight and print a string */
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" black/blue           ");
grub_env_set ("color_highlight", "black/green");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" black/green          ");
grub_env_set ("color_highlight", "black/cyan");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" black/cyan           ");
grub_env_set ("color_highlight", "black/red");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" black/red            \n");
grub_env_set ("color_highlight", "black/magenta");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" black/magenta        ");
grub_env_set ("color_highlight", "black/brown");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" black/brown          ");
grub_env_set ("color_highlight", "black/light-gray");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" black/light-gray     \n");
/* blue highlight */
grub_env_set ("color_highlight", "blue/green");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" blue/green           ");
grub_env_set ("color_highlight", "blue/cyan");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" blue/cyan           ");
grub_env_set ("color_highlight", "blue/red");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" blue/red            \n");
grub_env_set ("color_highlight", "blue/magenta");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" blue/magenta        ");
grub_env_set ("color_highlight", "blue/brown");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" blue/brown          ");
grub_env_set ("color_highlight", "blue/light-gray");
grub_setcolorstate (GRUB_TERM_COLOR_HIGHLIGHT);
grub_printf (" blue/light-gray     \n");
/* revert to original color_normal */
grub_env_set ("color_normal", color_normal);
grub_setcolorstate (GRUB_TERM_COLOR_NORMAL);
grub_printf ("\n");
/* and exit */
return err;
}
static grub_extcmd_t cmd;
GRUB_MOD_INIT(colortest)
{
    cmd = grub_register_extcmd ("colortest", grub_cmd_colortest,
                               GRUB_COMMAND_FLAG_BOTH,
                               "[-c]",
                               /* the _N construct is for message localization */
                               N_("Test supported color combinations."),
                               options);
}
GRUB_MOD_FINI(colortest)
{
    grub_unregister_extcmd (cmd);
}

```

Now that the C code is written, how can we integrate *colortest* into the existing GRUB2 build system so that *colortest.mod* is automatically built when we enter *make*. at the GRUB2 build top level directory? It turns out to be long-winded but easy. Just add the following lines to *../conf/common.mk*:

```
# For colortest.mod.
pkglib_MODULES += colortest.mod
colortest_mod_SOURCES = commands/colortest.c
clean-module-colortest.mod.1:
    rm -f colortest.mod mod-colortest.o mod-colortest.c pre-colortest.o colortest_mod-
commands_colortest.o und-colortest.lst
CLEAN_MODULE_TARGETS += clean-module-colortest.mod.1
clean-module-colortest.mod-symbol.1:
    rm -f def-colortest.lst
CLEAN_MODULE_TARGETS += clean-module-colortest.mod-symbol.1
DEFSYMFIES += def-colortest.lst
mostlyclean-module-colortest.mod.1:
    rm -f colortest_mod-commands_colortest.d
MOSTLYCLEAN_MODULE_TARGETS += mostlyclean-module-colortest.mod.1
UNDSYMFIES += und-colortest.lst
ifeq ($(TARGET_APPLE_CC),1)
colortest.mod: pre-colortest.o mod-colortest.o $(TARGET_OBJ2ELF)
    -rm -f $@
    $(TARGET_CC) $(colortest_mod_LDFLAGS) $(TARGET_LDFLAGS) -Wl,-r,-d -o $@ pre-colort
est.o mod-colortest.o
    if test ! -z "$(TARGET_OBJ2ELF)"; then ./$(TARGET_OBJ2ELF) $@ || (rm -f $@; exit 1)
; fi
    $(STRIP) --strip-unneeded -K grub_mod_init -K grub_mod_fini -K _grub_mod_init -K _grub_mod_fini -R .note -R .comment $@
else
colortest.mod: pre-colortest.o mod-colortest.o $(TARGET_OBJ2ELF)
    -rm -f $@
    -rm -f $@.bin
    $(TARGET_CC) $(colortest_mod_LDFLAGS) $(TARGET_LDFLAGS) -Wl,-r,-d -o $@.bin pre-co
lortest.o mod-colortest.o
    $(OBJCONV) -f$(TARGET_MODULE_FORMAT) -nr:_grub_mod_init:grub_mod_init -nr:_grub_mo
d_fini:grub_mod_fini -wd1106 -nu -nd $@.bin $@
    -rm -f $@.bin
endif
pre-colortest.o: $(colortest_mod_DEPENDENCIES) colortest_mod-commands_colortest.o
    -rm -f $@
    $(TARGET_CC) $(colortest_mod_LDFLAGS) $(TARGET_LDFLAGS) -Wl,-r,-d -o $@ colortest_
mod-commands_colortest.o
mod-colortest.o: mod-colortest.c
    $(TARGET_CC) $(TARGET_CPPFLAGS) $(TARGET_CFLAGS) $(colortest_mod_CFLAGS) -c -o $@
$<
mod-colortest.c: $(builddir)/moddep.lst $(srcdir)/genmodsrc.sh
    sh $(srcdir)/genmodsrc.sh 'colortest' $< > $@ || (rm -f $@; exit 1)
ifeq ($(TARGET_APPLE_CC),1)
def-colortest.lst: pre-colortest.o
    $(NM) -g --defined-only -P -p $< | sed 's/^\\([^\n]*\\).*/\\1 colortest/' > $@
else
def-colortest.lst: pre-colortest.o
    $(NM) -g -P -p $< | grep -E '^[_a-zA-Z0-9_]* [TDS]' | sed 's/^\\([^\n]*\\).*/\\1 color
test/' > $@
endif
und-colortest.lst: pre-colortest.o
    echo 'colortest' > $@
    $(NM) -u -P -p $< | cut -f1 -d' ' >> $@
colortest_mod-commands_colortest.o: commands/colortest.c $(commands/colortest.c_DEPENDENCI
ES)
    $(TARGET_CC) -Icommands -I$(srcdir)/commands $(TARGET_CPPFLAGS) $(TARGET_CFLAGS)
$(colortest_mod_CFLAGS) -MD -c -o $@ $<
    -include colortest_mod-commands_colortest.d
clean-module-colortest_mod-commands_colortest-extra.1:
```

```

rm -f cmd-colortest_mod-commands_colortest.lst fs-colortest_mod-commands_colortest.
lst partmap-colortest_mod-commands_colortest.lst handler-colortest_mod-commands_colortest.
lst parttool-colortest_mod-commands_colortest.lst video-colortest_mod-commands_colortest.l
st terminal-colortest_mod-commands_colortest.lst
CLEAN_MODULE_TARGETS += clean-module-colortest_mod-commands_colortest-extra.1
COMMANDFILES += cmd-colortest_mod-commands_colortest.lst
FSFILES += fs-colortest_mod-commands_colortest.lst
PARTTOOLFILES += parttool-colortest_mod-commands_colortest.lst
PARTMAPFILES += partmap-colortest_mod-commands_colortest.lst
HANDLERFILES += handler-colortest_mod-commands_colortest.lst
TERMINALFILES += terminal-colortest_mod-commands_colortest.lst
VIDEOFILES += video-colortest_mod-commands_colortest.lst
cmd-colortest_mod-commands_colortest.lst: commands/colortest.c $(commands/colortest.c_DEPE
NDENCIES) gencmdlist.sh
        set -e;           $(TARGET_CC) -Icommands -I$(srcdir)/commands $(TARGET_CPPFLAGS)
        $(TARGET_CFLAGS) $(colortest_mod_CFLAGS) -E $<          | sh $(srcdir)/gencmdlist.sh color
test > $@ || (rm -f $@; exit 1)
fs-colortest_mod-commands_colortest.lst: commands/colortest.c $(commands/colortest.c_DEPEND
ENCIES) genfslist.sh
        set -e;           $(TARGET_CC) -Icommands -I$(srcdir)/commands $(TARGET_CPPFLAGS)
        $(TARGET_CFLAGS) $(colortest_mod_CFLAGS) -E $<          | sh $(srcdir)/genfslist.sh color
test > $@ || (rm -f $@; exit 1)
parttool-colortest_mod-commands_colortest.lst: commands/colortest.c $(commands/colortest.c_
DEPENDENCIES) genparttoollist.sh
        set -e;           $(TARGET_CC) -Icommands -I$(srcdir)/commands $(TARGET_CPPFLAGS)
        $(TARGET_CFLAGS) $(colortest_mod_CFLAGS) -E $<          | sh $(srcdir)/genparttoollist.sh
color > $@ || (rm -f $@; exit 1)
partmap-colortest_mod-commands_colortest.lst: commands/colortest.c $(commands/colortest.c_
DEPENDENCIES) genpartmaplist.sh
        set -e;           $(TARGET_CC) -Icommands -I$(srcdir)/commands $(TARGET_CPPFLAGS)
        $(TARGET_CFLAGS) $(colortest_mod_CFLAGS) -E $<          | sh $(srcdir)/genpartmaplist.sh c
olor > $@ || (rm -f $@; exit 1)
handler-colortest_mod-commands_colortest.lst: commands/colortest.c $(commands/colortest.c_
DEPENDENCIES) genhandlerlist.sh
        set -e;           $(TARGET_CC) -Icommands -I$(srcdir)/commands $(TARGET_CPPFLAGS)
        $(TARGET_CFLAGS) $(colortest_mod_CFLAGS) -E $<          | sh $(srcdir)/genhandlerlist.sh c
olor > $@ || (rm -f $@; exit 1)
terminal-colortest_mod-commands_colortest.lst: commands/colortest.c $(commands/colortest.c_
DEPENDENCIES) genterminallist.sh
        set -e;           $(TARGET_CC) -Icommands -I$(srcdir)/commands $(TARGET_CPPFLAGS)
        $(TARGET_CFLAGS) $(colortest_mod_CFLAGS) -E $<          | sh $(srcdir)/genterminallist.sh
color > $@ || (rm -f $@; exit 1)
video-colortest_mod-commands_colortest.lst: commands/colortest.c $(commands/colortest.c_DE
PENDENCIES) genvideolist.sh
        set -e;           $(TARGET_CC) -Icommands -I$(srcdir)/commands $(TARGET_CPPFLAGS)
        $(TARGET_CFLAGS) $(colortest_mod_CFLAGS) -E $<          | sh $(srcdir)/genvideolist.sh col
or > $@ || (rm -f $@; exit 1)
colortest_mod_CFLAGS = $(COMMON_CFLAGS)
colortest_mod_LDFLAGS = $(COMMON_LDFLAGS)

```

Most of the above is boilerplate. I simply copied it from an existing target such as *tga* in *common.mk* and changed the target name to *colortest*. If you now execute *make* from the GRUB2 build toplevel directory, it should contain the module *colortest.mod* after the build completes if the build was successful.

Here is a detailed listing of the commands used to build *colortest.mod*:

```

gcc -Icommands -I./commands -nostdinc -isystem /usr/lib/gcc/x86_64-redhat-linux/4.4.4/include \
     -I./include -I. -I./include -Wall -W -Os -DGRUB_MACHINE_PCBIOS=1 -Wall -W -Wshadow \
     -Wpointer-arith -Wmissing-prototypes -Wundef -Wstrict-prototypes -g -falign-jumps=1 \
     -falign-loops=1 -falign-functions=1 -mno-mmx -mno-sse -mno-sse2 -mno-3dnow \

```

```
-fno-dwarf2-cfi-asn -m32 -fno-stack-protector -mno-stack-arg-probe -Werror -fno-build
tin -mrtd \
-mregparm=3 -m32 -MD -c -o colortest_mod-commands_colortest.o commands/colortest.c
rm -f pre-colortest.o
gcc -m32 -nostdlib -m32 -Wl,--build-id=none -Wl,-r,-d -o pre-colortest.o colortest_mod-com
mands_colortest.o
nm -g --defined-only -P -p pre-colortest.o | sed 's/^(\[^ ]*\).*/\1 colortest/' > def-co
rtest.lst
echo 'colortest' > und-colortest.lst
nm -u -P -p pre-colortest.o | cut -f1 -d' ' >> und-colortest.lst
```

As you can see below, a 32-bit ELF relocatable binary is produced.

```
file colortest.mod
colortest.mod: ELF 32-bit LSB relocatable, Intel 80386, version 1 (SYSV), not stripped
#
```

There are a number of ways to test and verify the new *colortest* module. You can copy *colortest.mod* into */boot/grub*, modify */boot/grub/command.lst* to add the following entry:

```
colortest: colortest
```

and reboot your system to the GRUB2 command prompt. From there you can load the *colortest* module using *insmod* and invoke the *grub_colortest* function by entering the command *colortest* or *colortest -c* at the *grub>* prompt. A drawback to this method is that it requires you to reboot your system every time you make a change to the *colortest* source code or build environment.

If you want to avoid having to repeatedly reboot your system you can test the functionality of *colortest* using the GRUB2 userspace emulator *grub-emu*. Note that *grub-emu* is not used for system booting; it is a standalone utility which you can run from the command line after you boot your system into GNU/Linux. It emulates, as best it can but with some limitations, the behavior of the GRUB2 command line.

To build *grub-emu* you need to have both the *ncurses-libs* and *ncurses-devel* packages installed. I did not find much documentation on how to build *grub-emu* but discovered how to do so mostly by trial and error. In the end it turned out to be quite simple.

```
./make clean
./configure --platform=emu
./make
./grub-emu
```

This should get you to the *grub-emu* prompt after a few warning messages are displayed. These warning messages are normal. Here is what *grub-emu* outputs when it is invoked with the *-h* option:

```
# ./grub-emu -h
Usage: ./grub-emu [OPTION]...
GRUB emulator.

-r, --root-device=DEV      use DEV as the root device [default=guessed]
-m, --device-map=FILE      use FILE as the device map [default=/boot/grub/device.map]
-d, --directory=DIR        use GRUB files in the directory DIR [default=/boot/grub]
```

```

-v, --verbose      print verbose messages
-H, --hold[=SECONDS]  wait until a debugger will attach
-h, --help        display this message and exit
-V, --version     print version information and exit

Report bugs to .
#

```

By the way, enter *exit* or *reboot* to exit *grub-emu*. Notice that no modules such as *normal.mod* or *colortest.mod* are built when building the GRUB2 emulator. That is because *insmod* does not work in *grub-emu* and therefore building the modules is unnecessary. The *insmod* command is there but it cannot do anything. If you try to use *insmod* to load a module from /boot/grub, it will respond with an *invalid arch independent ELF magic* error message. Modules are either built-in during the build process or are not available. At this time, if you try to execute *colortest* from within *grub-emu* you will get a *error: unknown command 'colortest'* error message.

To get *grub-emu* to recognize the *colortest* command a number of files need to be modified and *grub-emu* rebuilt. First, add the following two lines to *grub-emu-init.c*:

```

grub_colortest_init ();
grub_colortest_fini ();

```

The first line should be added to the *grub_init_all* procedure and the second line to the *grub_fini_all* procedure. Two suitable function prototypes, modeled on the other existing function prototypes, should be added to *grub-emu-init.h* also. The purpose of *grub_init_all* and *grub_fini_all* (see .../util/grub-emu.c) is to load the specified modules at *grub-emu* startup and unload them when *grub-emu* is terminating.

Next you need to modify *./conf/any-emu.mk* to add support for linking *colortest* into *grub-emu* when building *grun-emu*. At a minimum add *commands/colortest.c* to the *grub_emu_SOURCES* list. While not absolutely required, I would add *grub_emu-commands_colortest.o* and *grub_emu-commands_colortest.d* to the list of files to be removed by the cleanup directives, e.g. *clean-utility-grub-emu.1*, etc. Finally, we need to tell *any-emu.mk* how to compile *colortest.o* for inclusion in *grub-emu* by adding the following directives to the file:

```

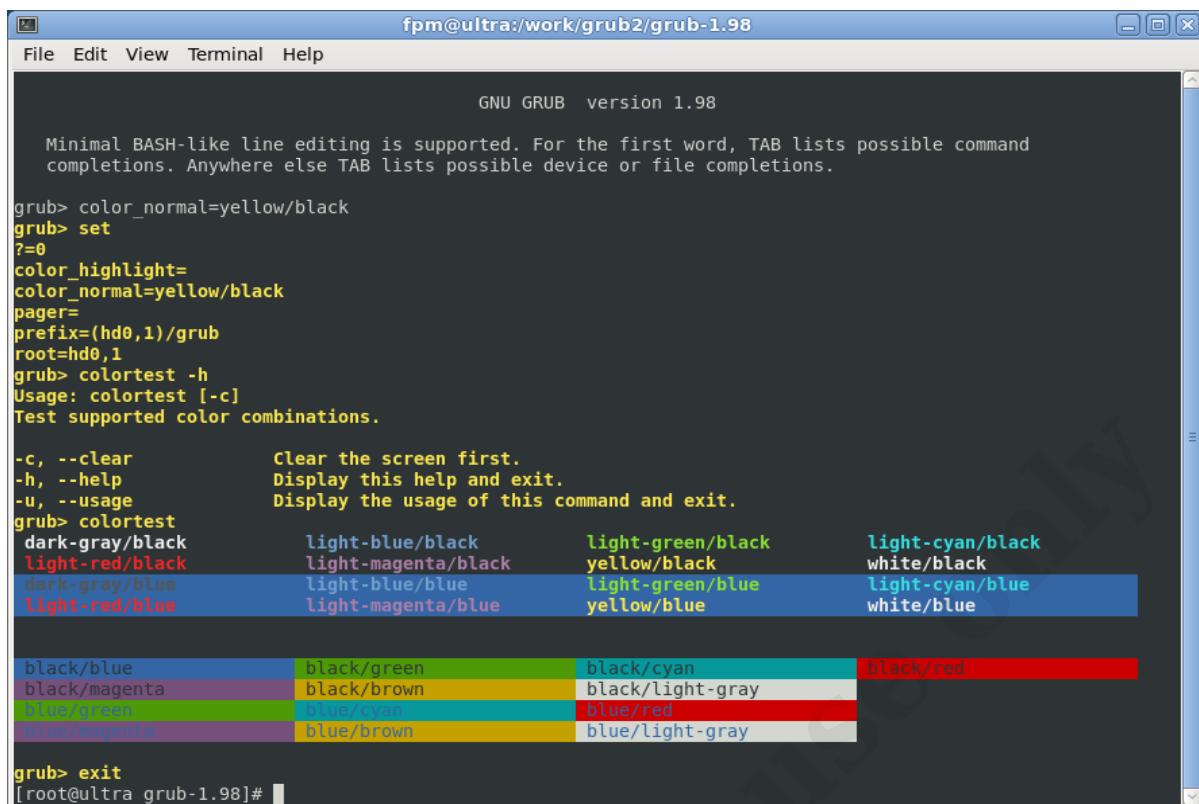
grub_emu-commands_colortest.o: commands/colortest.c $(commands/colortest.c_DEPENDENCIES)
    $(CC) -Icommands -I$(srcdir)/commands $(CPPFLAGS) $(CFLAGS) -DGRUB_UTIL=1 $(grub_e
mu_CFLAGS) -MD -c -o $@ $<
    -include grub_emu-commands_colortest.d

```

After rebuilding *grub-emu*, the *colortest* command should be listed when you enter the *help* command.

Here is a screenshot of *grub-emu* displaying the results of the *colortest* command.

GRUB2 Modules



The screenshot shows a terminal window titled "fpm@ultra:/work/grub2/grub-1.98". The window contains the following text:

```
GNU GRUB version 1.98

Minimal BASH-like line editing is supported. For the first word, TAB lists possible command completions. Anywhere else TAB lists possible device or file completions.

grub> color_normal=yellow/black
grub> set
?=0
color_highlight=
color_normal=yellow/black
pager=
prefix=(hd0,1)/grub
root=hd0,1
grub> colortest -h
Usage: colortest [-c]
Test supported color combinations.

-c, --clear      Clear the screen first.
-h, --help       Display this help and exit.
-u, --usage      Display the usage of this command and exit.
grub> colortest
dark-gray/black   light-blue/black    light-green/black  light-cyan/black
light-red/black   light-magenta/black yellow/black      white/black
dark-gray/blue    light-blue/blue     light-green/blue  light-cyan/blue
light-red/blue   light-magenta/blue  yellow/blue      white/blue

black/blue        black/green       black/cyan       black/red
black/magenta    black/brown      black/light-gray blue/red
blue/green       blue/cyan       blue/light-gray
blue/magenta    blue/brown      blue/cyan

grub> exit
[root@ultra grub-1.98]#
```

Another way to safely test a new module is to host GRUB2 on an IA32 processor emulator such as [Qemu](#) or [Bochs](#). However, I am not going to cover those methods in this post. If I find the time I will cover them in a separate post in the near future.

I am going to stop writing now. Hopefully, after reading this post, you have a better understanding of how to write, integrate and test a GRUB2 module. Please let me know if there is other pertinent information I should add to this post which would assist readers in understanding the relevant issues.

P.S. I built and tested the examples included in this post on X64 platform running Fedora 13.