Recursive wildcards in GNU make?

Asked 14 years, 7 months ago Modified 1 year, 1 month ago Viewed 86k times

121

It's been a while since I've used `make`, so bear with me...

I've got a directory, `flac`, containing .FLAC files. I've got a corresponding directory, `mp3` containing MP3 files. If a FLAC file is newer than the corresponding MP3 file (or the corresponding MP3 file doesn't exist), then I want to run a bunch of commands to convert the FLAC file to an MP3 file, and copy the tags across.

The kicker: I need to search the `flac` directory recursively, and create corresponding subdirectories in the `mp3` directory. The directories and files can have spaces in the names, and are named in UTF-8.

And I want to use `make` to drive this.

makefile gnu-make

Share Edit Follow Flag

edited Mar 20, 2010 at 13:56
P Shved
98.7k 19 127 167

asked Mar 20, 2010 at 13:28
Roger Lipscombe
91.3k 59 252 394

1 Any reason for selecting make for this purpose? I'd have thought writing a bash script would be simpler – anon Mar 20, 2010 at 13:37

4 @Neil, make's concept as pattern-based file system transformation is the best way to approach the original problem. Perhaps implementations of this approach have its limitations, but `make` is closer to implementing it than bare `bash`. – P Shved Mar 20, 2010 at 13:55

1 @Pavel Well, a `sh` script that walks through the list of flac files (`find | while read flacname`), makes a `mp3name` from that, runs "mkdir -p" on the `dirname "$mp3name"`, and then, `if [ "$flacfile" -nt "$mp3file"]` converts `"$flacname"` into `"$mp3name"` is not really magic. The only feature you are actually losing compared to a `make` based solution is the possibility to run `N` file conversions processes in parallel with `make -jN`. – ndim Mar 20, 2010 at 15:45

4 @ndim That's the first time I have ever heard make's syntax be described as "nice" :-) – anon Mar 20, 2010 at 16:05

2 Using `make` and having spaces in file names are contradictory requirements. Use a tool appropriate for the problem domain. – Jens Aug 15, 2013 at 17:45

7 Answers

Sorted by: Highest score (default)

143

I would try something along these lines

```
FLAC_FILES = $(shell find flac/ -type f -name '*.flac')
MP3_FILES = $(patsubst flac/%.flac, mp3/%.mp3, $(FLAC_FILES))

.PHONY: all
all: $(MP3_FILES)

mp3/%.mp3: flac/%.flac
	@mkdir -p "$(@D)"
	@echo convert "$<" to "$@"
```

A couple of quick notes for `make` beginners:

- The `@` in front of the commands prevents `make` from printing the command before actually running it.
- `$(@D)` is the directory part of the target file name (`$@`)
- Make sure that the lines with shell commands in them start with a tab, not with spaces.

Even if this should handle all UTF-8 characters and stuff, it will fail at spaces in file or directory names, as `make` uses spaces to separate stuff in the makefiles and I am not aware of a way to work around that. So that leaves you with just a shell script, I am afraid :-/

Share Edit Follow Flag

edited Jun 19, 2012 at 12:58
zrajm
1,403 1 13 21

answered Mar 20, 2010 at 13:35
ndim
37.5k 12 49 58

This is where I was going...fast fingers for the win. Though it looks like you may be able to do something clever with `vpath`. Must study that one of these days.
– dmckee --- ex-moderator kitten Mar 20, 2010 at 13:41

2 Doesn't appear to work when the directories have spaces in the names. – Roger Lipscombe Mar 20, 2010 at 13:45

1 Didn't realize that I'd have to shell out to `find` to get the names recursively... – Roger Lipscombe Mar 20, 2010 at 13:47

1 @PaulKonova: Run `make -jN`. For `N` use the number of conversions which `make` should run in parallel. Caution: Running `make -j` without an `N` will start all conversion processes at once in parallel which might be equivalent to a fork bomb. – ndim Nov 19, 2013 at 20:40

1 @Adrian: The `.PHONY: all` line tells make that the recipe for the `all` target is to be executed even if there is a file called `all` newer than all the `$(MP3_FILES)`. – ndim Apr 26, 2018 at 11:27

93

You can define your own recursive wildcard function like this:

```
rwildcard=$(foreach d,$(wildcard $(1:=/*)),$(call rwildcard,$d,$2) $(filter $(subst
*,%,$2),$d))
```

The first parameter (`$1`) is a list of directories, and the second (`$2`) is a list of patterns you want to match.

Examples:

To find all the C files in the current directory:

```
$(call rwildcard,.,*.c)
```

To find all the `.c` and `.h` files in `src` :

```
$(call rwildcard,src,*.c *.h)
```

This function is based on the implementation from this article, with a few improvements.

Share Edit Follow Flag edited Oct 10, 2019 at 16:37 HolyBlackCat **93.7k** 12 164 255 answered Aug 15, 2013 at 17:37 user1896626

This doesn't seem to work for me. I've copied the exact function and it still won't look recursively. – Jeroen Nov 5, 2013 at 19:38

5 I am using GNU Make 3.81, and it seems to work for me. It won't work if any of the filenames have spaces in them, though. Note that the filenames it returns have paths relative to the current directory, even if you are only listing files in a subdirectory. – user1896626 Nov 13, 2013 at 6:53

3 This is truly an example, that `make` is a Turing Tar Pit (see here: yosefk.com/blog/fun-at-the-turing-tar-pit.html). It is not even that hard, but one has to read this: gnu.org/software/make/manual/html_node/Call-Function.html and then "understand recurrence". YOU had to write this recursively, in the verbatim sense; it's not the everyday understanding of "automatically include stuff from subdirs". It's actual RECURRENCE. But remember - "To understand recurrence, you have to understand recurrence". – Tomasz Gandor Aug 6, 2014 at 22:13

1 @TomaszGandor You don't have to understand recurrence. You have to understand recursion and in order to do that you must first understand recursion. – user1129682 May 7, 2019 at 15:19

1 Despite the tar pit, this is the right answer because it is portable and does not depend on shell commands. – Kenn Sebesta Feb 26, 2022 at 23:11

6

If you're using Bash 4.x, you can use a new globbing option, for example:

```
SHELL:=/bin/bash -O globstar
list:
  @echo Flac: $(shell ls flac/**/*.flac)
  @echo MP3: $(shell ls mp3/**/*.mp3)
```

This kind of recursive wildcard can find all the files of your interest (*.flac*, *.mp3* or whatever). O

Share Edit Follow Flag edited Nov 30, 2017 at 13:02 answered May 10, 2015 at 18:01

kenorb
165k 94 702 766

3 To me, even just $(wildcard flac/**/*.flac) seems to work. OS X, Gnu Make 3.81 – akauppi May 13, 2015 at 11:10

2 I tried $(wildcard ./**/*.py) and it behaved the same as $(wildcard ./*/*.py). I don't think make actually supports **, and it just doesn't fail when you use two *s next to each other. – lahwran Nov 22, 2016 at 21:06

@lahwran It should when you invoking commands via Bash shell and you've enabled *globstar* option. Maybe you're not using GNU make or something else. You may also try this syntax instead. Check the comments for some suggestions. Otherwise it's a thing for the new question. – kenorb Nov 23, 2016 at 0:20

@kenorb no no, I didn't even try your thing because I wanted to avoid shell invocation for this particular thing. I was using akauppi's suggested thing. The thing I went with looked like larskholte's answer, though I got it from somewhere else because the comments here said this one was subtly broken. shrug :) – lahwran Nov 23, 2016 at 1:12

1 @lahwran In this case `**` won't work, because the extended globbing is a bash/zsh thing. – kenorb Nov 23, 2016 at 11:06

2

Here's a Python script I quickly hacked together to solve the original problem: keep a compressed copy of a music library. The script will convert .m4a files (assumed to be ALAC) to AAC format, unless the AAC file already exists and is newer than the ALAC file. MP3 files in the library will be linked, since they are already compressed.

Just beware that aborting the script (ctrl-c) will leave behind a half-converted file.

I originally also wanted to write a Makefile to handle this, but since it cannot handle spaces in filenames (see the accepted answer) and because writing a bash script is guaranteed to put in me in a world of pain, Python it is. It's fairly straightforward and short, and thus should be easy to tweak to your needs.

```
import glob
import os
import subprocess


UNCOMPRESSED_DIR = 'Music'
COMPRESSED = 'compressed_'

UNCOMPRESSED_EXTS = ('m4a', )   # files to convert to lossy format
LINK_EXTS = ('mp3', )           # files to link instead of convert


for root, dirs, files in os.walk(UNCOMPRESSED_DIR):
    out_root = COMPRESSED + root
    if not os.path.exists(out_root):
        os.mkdir(out_root)
    for file in files:
        file_path = os.path.join(root, file)
```

```
                print('Linking {}'.format(file_path))
                link_source = os.path.relpath(file_path, out_root)
                os.symlink(link_source, COMPRESSED + file_path)
            continue
        if ext[1:] not in UNCOMPRESSED_EXTS:
            print('Skipping {}'.format(file_path))
            continue
        out_file_path = COMPRESSED + file_path
        if (os.path.exists(out_file_path)
            and os.path.getctime(out_file_path) > os.path.getctime(file_path)):
            print('Up to date: {}'.format(file_path))
            continue
        print('Converting {}'.format(file_path))
        subprocess.call(['ffmpeg', '-y', '-i', file_path,
                         '-c:a', 'libfdk_aac', '-vbr', '4',
                         out_file_path])
```

Of course, this can be enhanced to perform the encoding in parallel. That is left as an exercise to the reader ;-)

Share Edit Follow Flag edited Jun 6, 2017 at 7:54 answered Dec 25, 2016 at 19:38

Brecht Machiels
3,380 4 28 40

2

FWIW, I've used something like this in a *Makefile*:

```
RECURSIVE_MANIFEST = `find . -type f -print`
```

The example above will search from the current directory ('.') for all "plain files" ('-*type f*') and set the `RECURSIVE_MANIFEST` make variable to every file it finds. You can then use pattern substitutions to reduce this list, or alternatively, supply more arguments into *find* to narrow what it returns. See the man page for *find*.

Share Edit Follow Flag answered Dec 3, 2011 at 20:09

Michael Shebanow
49 2

2

My solution is based on the one above, uses `sed` instead of `patsubst` to mangle the output of `find` AND escape the spaces.

Going from flac/ to ogg/

```
OGGS = $(shell find flac -type f -name "*.flac" | sed 's/ /\\
/g;s/flac\//ogg\//;s/\.flac/\.ogg/' )
```

Caveats:

1. Still barfs if there are semi-colons in the filename, but they're pretty rare.
2. The $(@D) trick won't work (outputs gibberish), but oggenc creates directories for you!

Share Edit Follow Flag

edited Jul 19, 2015 at 12:06
Community Bot
1 1

answered Jan 15, 2014 at 18:52
user3199485
29 1

0

To find files recursively without resorting to external dependencies like `find`, you can use functions. Then use the result as in the other answer to convert the files.

```
rwildcard=$(wildcard $1) $(foreach d,$1,$(call rwildcard,$(addsuffix /$(notdir
$d),$(wildcard $(dir $d)*))))

FLAC_FILES = $(call rwildcard,flac/*.flac)
MP3_FILES = $(patsubst flac/%.flac, mp3/%.mp3, $(FLAC_FILES))

.PHONY: all
all: $(MP3_FILES)

mp3/%.mp3: flac/%.flac
	@mkdir -p "$(@D)"
	@echo convert "$<" to "$@"
```

Share Edit Follow Flag

answered Aug 25, 2023 at 11:42
Marko Kohtala
892 7 18

1 See github.com/markpiffer/gmtt#call-wildcard-reclist-of-globs for a beefed up version of recursive wildcards – Vroomfondel Aug 28, 2023 at 8:35