NAME

File::GlobMapper - Extend File Glob to Allow Input and Output Files

SYNOPSIS

```
use File::GlobMapper qw( globmap );

my $aref = globmap $input => $output
    or die $File::GlobMapper::Error ;

my $gm = new File::GlobMapper $input => $output
    or die $File::GlobMapper::Error ;
```

DESCRIPTION

This module needs Perl5.005 or better.

This module takes the existing `File::Glob` module as a starting point and extends it to allow new filenames to be derived from the files matched by `File::Glob`.

This can be useful when carrying out batch operations on multiple files that have both an input filename and output filename and the output file can be derived from the input filename. Examples of operations where this can be useful include, file renaming, file copying and file compression.

Behind The Scenes

To help explain what `File::GlobMapper` does, consider what code you would write if you wanted to rename all files in the current directory that ended in `.tar.gz` to `.tgz`. So say these files are in the current directory

```
alpha.tar.gz
beta.tar.gz
gamma.tar.gz
```

and they need renamed to this

```
alpha.tgz
beta.tgz
gamma.tgz
```

Below is a possible implementation of a script to carry out the rename (error cases have been omitted)

```
foreach my $old ( glob "*.tar.gz" )
{
    my $new = $old;
    $new =~ s#(.*)\.tar\.gz$#$1.tgz# ;

    rename $old => $new
        or die "Cannot rename '$old' to '$new': $!\n;
}
```

Notice that a file glob pattern `*.tar.gz` was used to match the `.tar.gz` files, then a fairly similar regular expression was used in the substitute to allow the new filename to be created.

Given that the file glob is just a cut-down regular expression and that it has already done a lot of the hard work in pattern matching the filenames, wouldn't it be handy to be able to use the patterns in the fileglob to drive the new filename?

Well, that's *exactly* what `File::GlobMapper` does.

Here is same snippet of code rewritten using `globmap`

```
for my $pair (globmap '<*.tar.gz>' => '<#1.tgz>' )
{
    my ($from, $to) = @$pair;
    rename $from => $to
        or die "Cannot rename '$old' to '$new': $!\n;
}
```

So how does it work?

Behind the scenes the `globmap` function does a combination of a file glob to match existing filenames followed by a substitute to create the new filenames.

Notice how both parameters to `globmap` are strings that are delimited by <>. This is done to make them look more like file globs - it is just syntactic sugar, but it can be handy when you want the strings to be visually distinctive. The enclosing <> are optional, so you don't have to use them - in fact the first thing globmap will do is remove these delimiters if they are present.

The first parameter to `globmap`, `*.tar.gz`, is an *Input File Glob*. Once the enclosing "< ... >" is removed, this is passed (more or less) unchanged to `File::Glob` to carry out a file match.

Next the fileglob `*.tar.gz` is transformed behind the scenes into a full Perl regular expression, with the additional step of wrapping each transformed wildcard metacharacter sequence in parenthesis.

In this case the input fileglob `*.tar.gz` will be transformed into this Perl regular expression

```
([^/]*)\.tar\.gz
```

Wrapping with parenthesis allows the wildcard parts of the Input File Glob to be referenced by the second parameter to `globmap`, `#1.tgz`, the *Output File Glob*. This parameter operates just like the replacement part of a substitute command. The difference is that the `#1` syntax is used to reference sub-patterns matched in the input fileglob, rather than the `$1` syntax that is used with perl regular expressions. In this case `#1` is used to refer to the text matched by the `*` in the Input File Glob. This makes it easier to use this module where the parameters to `globmap` are typed at the command line.

The final step involves passing each filename matched by the `*.tar.gz` file glob through the derived Perl regular expression in turn and expanding the output fileglob using it.

The end result of all this is a list of pairs of filenames. By default that is what is returned by `globmap`. In this example the data structure returned will look like this

```
( ['alpha.tar.gz' => 'alpha.tgz'],
  ['beta.tar.gz'  => 'beta.tgz' ],
  ['gamma.tar.gz' => 'gamma.tgz']
)
```

Each pair is an array reference with two elements - namely the *from* filename, that `File::Glob` has matched, and a *to* filename that is derived from the *from* filename.

Limitations

`File::GlobMapper` has been kept simple deliberately, so it isn't intended to solve all filename mapping operations. Under the hood `File::Glob` (or for older versions of Perl, `File::BSDGlob`) is used to match the files, so you will never have the flexibility of full Perl regular expression.

Input File Glob

The syntax for an Input FileGlob is identical to `File::Glob`, except for the following

1. No nested {}
2. Whitespace does not delimit fileglobs.
3. The use of parenthesis can be used to capture parts of the input filename.
4. If an Input glob matches the same file more than once, only the first will be used.

The syntax

~

~user

.

Matches a literal '.'. Equivalent to the Perl regular expression

```
\.
```

- Matches zero or more characters, except '/'. Equivalent to the Perl regular expression

```
[^/]*
```

?

Matches zero or one character, except '/'. Equivalent to the Perl regular expression

```
[^/]?
```

****

Backslash is used, as usual, to escape the next character.

[]

Character class.

{,}

Alternation

()

Capturing parenthesis that work just like perl

Any other character it taken literally.

Output File Glob

The Output File Glob is a normal string, with 2 glob-like features.

The first is the '*' metacharacter. This will be replaced by the complete filename matched by the input file glob. So

```
*.c *.Z
```

The second is

Output FileGlobs take the

"*"

The "*" character will be replaced with the complete input filename.

#1

Patterns of the form /#\d/ will be replaced with the

Returned Data

EXAMPLES

A Rename script

Below is a simple "rename" script that uses `globmap` to determine the source and destination filenames.

```
use File::GlobMapper qw(globmap) ;
use File::Copy;

die "rename: Usage rename 'from' 'to'\n"
    unless @ARGV == 2 ;

my $fromGlob = shift @ARGV;
my $toGlob   = shift @ARGV;

my $pairs = globmap($fromGlob, $toGlob)
    or die $File::GlobMapper::Error;

for my $pair (@$pairs)
{
    my ($from, $to) = @$pair;
    move $from => $to ;
}
```

Here is an example that renames all c files to cpp.

```
$ rename '*.c' '#1.cpp'
```

A few example globmaps

Below are a few examples of globmaps

To copy all your .c file to a backup directory

```
'</my/home/*.c>'    '</my/backup/#1.c>'
```

If you want to compress all

```
'</my/home/*.[ch]>'    '<*.gz>'
```

To uncompress

```
'</my/home/*.[ch].gz>'    '</my/home/#1.#2>'
```

SEE ALSO

File::Glob

AUTHOR

The *File::GlobMapper* module was written by Paul Marquess, *pmqs@cpan.org*.

COPYRIGHT AND LICENSE