

COPYRIGHT NOTICE

This article is Copyright (c) Al Andersen, <http://www.alandersen.com/>

You are free to use this code in your programs providing you adhere to the most current version of the GNU General Public License (GPL), <http://www.gnu.org/licenses/gpl.html>

DISCLAIMERS AND CAVEATS

This code is provided, as is. I don't guarantee that it will work, and I don't guarantee that it will not destroy data on your hard disk. Use at your own risk! Backup important files!

This code requires that a command line version of PHP is installed on your computer.

This program requires that exiv2 be installed on your computer. Check your package manager or get it from <http://www.exiv2.org/>. Make sure it is in your executable path.

This code is for use on your local computer only. **Do not use this program on a server – it is not secure!**

RECURSIVE PHP FUNCTION FOR RENAMING FILES

In our last tutorial we wrote a function that lets us recursively review files in an entire directory tree. In this tutorial we will modify that function so we can rename specific file types. This is useful for cataloging our JPG or NEF files, giving them a unique datetime stamp for their file names.

Most cameras name their JPGs or NEFs with a common prefix and unique numeric ID. This ID will either reset itself to 1 every time we change our camera storage media, or it will cycle through the gamut of numbers to eventually roll-over back to the first number in the series.

While practical, the assigned file names are not very useful. If we take our photography seriously we'll eventually exceed the allowed number of unique file names and wind up with duplicates. And if we really take our photography seriously, we'll have no idea what the contents of a file called `DSC0925.JPG` may be, especially if we have an archive of 10,000+ images!

Therefore, it would be nice if we could come up with a simple method of storing our images that would let us easily find them at a later date.

One way would be to give all our files a descriptive name, such as `willie_the_wonder_cat_0001.jpg`. Though descriptive and providing up to 9999 different files of that name, imagine how much effort it would take for us to rename the 9999 images of willie the wonder cat that we downloaded from our camera storage card. Nope, this method isn't scalable. We need to come up with something easier, something that we can automate and involves hardly any effort at all.

How about we name our files by the dates and times contained in the image EXIF data? The file names should end up being unique and if we put the files in separate subdirectories named by year, month, date, and some short descriptive text, we would end up with a chronological archive of all our images, logically stored in directories that would hint at what the image contents might be, all done with minimal time and effort.

Here's how our archive will look, showing the files within one of our sub-directories.

```
2009
  20090101_new_years_day
    20090101_080124.jpg
    20090101_080129.jpg
    20090101_080132.jpg
    20090101_080215.jpg
  20090201_february_already
  20090704_fourth_of_july_fireworks
  20090704_fourth_of_july_parade
  20091225_christmas_photos
2010
  20090101_another_new_years_day
```

It's an easy system to implement and maintain. We can use any file browser to peruse the subdirectories and quickly locate the images we're looking for. If we have a bunch of unrelated files for a specific date we simply create several sub-directories for that date and group the files into the appropriate sub-directory. Best of all, it's scalable--it makes no difference how many years or files we add, the only limit being how much disk space we have.

Exiv2

So, how do we do the renaming of the files? Well, there are various options and PHP even has some built-in EXIF functions that would let us use it exclusively for this task. However, there are some gotcha's involved, the main one being that even with millisecond resolution of the times in the EXIF data, newer cameras have really fast frame rates that will cause problems when we rename our files. We need a way to ensure that we don't come up with duplicate file names, or worse yet, duplicate a filename and overwrite a file we already renamed! Because of this problem, we use a program called `exiv2`, a utility that's been thoroughly tested and has flexible command options that provide us with a lot of power and versatility when it comes to viewing or manipulating EXIF data in image files.

Be sure you have `exiv2` installed on your machine. Check your package management system. If you can't find it in there, go grab it at <http://www.exiv2.org/>. Once installed, type `exiv2 --help` at a command prompt to look at the available command options. We'll be using this variation of parameters for our renaming program:

```
exiv2 -F -t -r %Y%m%d_%H%M%S complete_pathname_to_file;
```

`-F` simply prevents `exiv2` from pausing program execution with "are you sure" types of prompts prior to renaming our files.

`-t` sets the file timestamp for our renaming action.

`-r` sets the timestamp mask. We will use `yyyymmddd_hhmmss`, so our files are stored chronologically by year, month, day, hour, minute, and finally second.

One of the things `exiv2` does is to automatically append a `_number` onto the filename if it creates a duplicate. Therefore, if we shot several files at a high frame rate and their timestamps are the same, `exiv2` will give us `yyyymmddd_hhmmss_1`, `yyyymmddd_hhmmss_2`, `yyyymmddd_hhmmss_3`, etc, and we don't have to worry about duplicates or overwritten files.

The Code

Before we start, we need to change our script into an executable program. To do this, we add the following line of code to the top of our script. It must be the first line of code with no empty spaces, line feeds, carriage returns, etc., preceding it:

```
#!/usr/bin/php
```

Your system might have the PHP CLI executable in a directory other than `/usr/bin`, or it may be named something else, like `php-cli`. Therefore, make sure you use the proper directory and executable names!

After you do this, save your file and change the permissions on it to make it executable. On linux, you do this by typing:

```
chmod 700 name_of_script.php
```

If you move your script into your home `bin` directory, you should be able to type the name of the script and have it execute without having to precede it with the `php` command.

Now that we have made the script executable, let's edit it to accommodate our renaming code.

Our new and improved recursive function is going to take three parameters:

```
function read_directory ($p_directory, $p_match, $p_suffix)
```

`$p_directory` = parameter containing the complete pathname to the directory we want to process.

`$p_match` = parameter containing the file extension that must be matched to trigger the renaming code.

`$p_suffix` = parameter containing a suffix that we might want to add to the filename.

We make a modification to our recursive function, replacing the `ELSE` part of the `if ($filetype == "dir")` statement with the following code.

```
else
{
    // We're looking for files only with the extension
    // passed in the $p_match parameter.
    if (preg_match ("/{ $p_match }$/i", $file))
    {
        // Do we have a suffix? If so, add it.
        $suffix = (empty ($p_suffix)) ? "" : "_{$p_suffix}";

        // *nix is a case-sensitive. Make it all lower-case.
        if (strtolower ($file) != $file)
        {
            $lower_case = strtolower ($file);
            rename ("{$d->path}/{$file}", "{$d->path}/{$lower_case}");
            $file = $lower_case;
        }

        // Use exiv2 to rename the files using the EXIF date timestamps.
        // Use "man exiv2" if you want to know what the command does.
        // Add the full path to exiv2 if necessary,
        // e.g., /usr/local/bin/exiv2
        $cmd = "exiv2 -F -t -r %Y%m%d_%H%M%S{$suffix} {$d->path}/{$file}";
```

```

        echo "\nRENAME: {$d->path}/{$file}";
        $result = shell_exec ($cmd);
    }
}

```

Let's go over the above code.

First, we need to ensure that we're only looking at filenames that match the files we want to rename. If doing JPGs, we send in JPG. If doing Nikon RAW files, we send in NEF. The `preg_match` function looks for the appropriate extension in each filename. If it finds it, the renaming code gets executed. If it doesn't, it falls through and nothing happens to that file.

We then look for a suffix parameter. The suffix can be anything we want. If the suffix is empty, nothing is added to the new filename. I usually pass my initials or my wife's first name as a file suffix. Our photos are stored together in the same off-disk archive. We can identify who took which photo by the filename suffix.

We then convert the filename to lower case. We do this to ensure that there are no problems on operating systems that allow mixed case file names (like linux). We try to do everything we can to preclude duplicate file names!

Finally, we get to the part where `exiv2` actually renames the file. We construct the filename format using the `-r` parameter, and tack on the `$suffix` we want, if any. We then pass the entire `exiv2` command string to our command shell using the `shell_exec` function.

That's it! We can rename several hundred files in just a couple of seconds by passing in the appropriate parameters to our recursive function.

Before we wrap this tutorial up, let's look at what it is we pass to our recursive function, starting with a bit of extra code that will prompt us for what we need to do should we forget what it is we must do.

Providing that we have the proper settings in our PHP.INI file (should be, by default), we can test `$argv` to see what "arguments" (parameters) we pass to our program and `$argc` for the number of arguments passed. We'll test `$argv`, which is an array. `$argv[0]` will always contain the name of our program (PHP arrays are zero-based). If `$argv` has no further elements then no arguments were passed and we should prompt ourselves for the requisite command syntax.

```

if (empty ($argv[1]))
{
    echo "\nSYNTAX: {$argv[0]}: directory_to_process [suffix]";
    echo "\n\tdirectory_to_process: MUST be a complete pathname!";
    echo "\n\t[suffix]: optional parameter, suffix for each new
filename.";
    echo "\n\n\ti.e., {$argv[0]} /home/mystuff/photos set1\n\n";
    exit();
}

```

We put any arguments we pass into some variables we can use. This is mostly for readability--we could simply use these arguments in our function calls. Please note that we don't test the order of the arguments. Therefore, make sure you pass the arguments into the program in the required order!

```

$pathname = $argv[1];
$suffix   = $argv[2];

```

Change to the directory we want to process.

```

chdir ($pathname);

```

Now process any JPG files we have there.

```
$extension = "jpg";  
read_directory ($pathname, $extension, $suffix);
```

Now go through the directory again and this time process any NEF files we may have.

```
$extension = "nef";  
read_directory ($pathname, $extension, $suffix);
```