User's Guide

Table of contents

1. Introduction

This *User's Guide* gives an introduction to the use of hdrff. It covers a few typical use-cases, but it is not a complete reference.

This guide assumes that you have installed hdrff and it's prerequisite packages as described in Installation. You should also have a working basic configuration (also described there).

If you don't want to setup the complete workflow, but just want to process a few images, you should nevertheless read the following sections to gain an understanding of the concepts. But instead of running hdrff, you would use hdrff4img which is also described in this section of this user's guide.

2. Modules

The central concept of hdrff is the *"module"*. A module performs a dedicated task during the workflow. A module should do one and only one thing (for all images processed during one run). For example, the copyFiles-module copies your images from your memory card to your hard-disk. The module copyFiles is typically the first module in every workflow.

You tell hdrff which modules it should process by passing them on the commandline, e.g.:

```
$ > hdrff copyFiles name2lc fixDate setRO raw2tif findGroups
```

Typically, you can guess what a module does from it's name. If you need details, you either have to read the documentation, or you just run `hdrff -H`.

You don't have to run all modules at the same time. The example given is equivalent to running

```
$ > hdrff copyFiles name2lc fixDate
$ > hdrff setRO raw2tif findGroups
```

or

```
$ > hdrff copyFiles
$ > hdrff name2lc fixDate setRO raw2tif
$ > hdrff findGroups
```

This feature lets you split up your workflow into separate phases. The section An Advanced Workflow below will use hdrff in multiple phases.

3. Module Groups

Passing all the modules on the commandline like in the example above is error-prone and complicated. To facilitate processing and to make life simple, hdrff also supports *"module-groups"*. A module-group is a list of modules. Module-groups can be nested, i.e. a module-group can contain other module-groups.

Hdrff comes with a number of predefined module-groups. The example above is equivalent to

```
$ > hdrff baseModules
```

You will see a complete list of the defined module-groups with the command `hdrff -H`.

Technically, module-groups are just environment variables. So you can define your own groups like this:

```
$ > export myModules="baseModules delInterFiles mv2archive"
$ > hdrff myModules
```

You don't have to define your groups on the commandline, you can also put the definition into your hdrff-configuration-file. Note that it is only a convention that the name of a module-group contains the string *"Modules"*. You can use any name, but be aware that `hdrff -H` only shows module-groups which follow the naming convention.

There is one special module-group: `MODULES`. This group contains all the modules hdrff should run if you don't pass a module or module-group on the commandline. If you don't like this behaviour, you could set `MODULES=noop`. Hdrff will then only run the no-op module noop.

4. A simple Single-Phase Workflow

The following example is a simple single phase workflow. Let's assume we are processing images in Nikon's raw-format (nef-files). We use the following `$HOME/.hdrff/hdrff.conf`:

```
: ${MEDIA:=/mnt/media}
: ${TARGET_DIR:=/data/images}
: ${IMG_EXT:=nef}
: ${IMG_PREFIX:=dsc_}
: ${IMG_DEPTH:=16}
: ${IMG_FIXCA:=1}
: ${IMG_ALIGN:=1}
: ${MODULES=enfuseModules makeHDR tmMantiuk tmFattal makeGIMP}
: ${PREVIEW_MODE=1}
: ${PREVIEW_SIZE=1024}
```

Now that everything is defined, we mount the memory card on `/mnt/media` and start hdrff:

```
$ > hdrff
```

You will see a lot of messages passing by. You can change this by either passing the `-q` option to hdrff or by setting `VERBOSE=0` in your configuration file.

Starting hdrff without arguments will execute all the modules defined in the `MODULES`-configuration variable. In our case, hdrff will do the following:

- Copy files from `$MEDIA` to `$TARGET_DIR/orig.`
- Rename files to lowercase.
- Set the filedate to the exifdate.
- Set the files in `$TARGET_DIR/orig` to read-only.
- Convert the files in `$TARGET_DIR/orig` to tif-files in `$TARGET_DIR/work`. A file named `dsc_1234.nef` will generate a file named `dsc_1234.tif`.
- Identify image-sequences. This step will create the file `$TARGET_DIR/work/hdr-list.txt`.
- Fix chromatic aberrations. This will generate a files named `ca_*.tif`, e.g. `ca_1234.tif` in `$TARGET_DIR/orig`. In case you use `IMG_FIXCA=0` the module will only create symlinks instead of files.
- Align images. Lets assume three images `dsc_1234.nef-dsc_1236.nef` constitute one sequence and images `dsc_1237.nef` and `dsc_1238.nef` are "single" images. The alignment-module alignImages will create three new files `ais_1234.tif-ais_1236.tif` and two symlinks `ais_1237.tif` and `ais1238.tif` symlinked to the respective ca_*-file.
- The module enfuseImages will merge the image sequences. Our example will create one file `enf_1234.tif` and two symlinks `enf_1237.tif` and `enf_1238.tif`.
- Next, makeHDR generates a "true" 32-bit HDR-file. The name is `hdr_1234.hdr`.
- The next two modules tonemap this HDR-file with two different algorithms (Mantiuk and Fattal). Thesse steps generate the files `mantiuk_1234.tif` and `fattal_1234.tif`.
- Finally, the module makeGIMP creates a file named `gimp_1234.xcf`. This is a three-layered file using `enf_1234.tif`, `mantiuk_1234.tif` and `fattal_1234.tif`.

At this point the workflow is almost finished. You could stick with either the file generated by enfuse or one of the tonemapped files. You could also start GIMP and fine-tune `gimp_1234.xcf`.

Although this simple workflow generates useful files from image-sequences, there are still some open ends. There are a lot of work-files left over together with the final files in the working directory. The next chapter will cover additional tasks in the workflow.

5. An advanced Workflow

The advanced workflow assumes the same configuration as above with only a few changes:

```
: ${MEDIA:=/mnt/media}
: ${TARGET_DIR:=/data/images}
: ${IMG_EXT:=nef}
: ${IMG_PREFIX:=dsc_}
: ${IMG_DEPTH:=16}
: ${IMG_FIXCA:=1}
: ${IMG_ALIGN:=1}
: ${phase1Modules=baseModules}
: ${phase2Modules=enfuseModules makeHDR tmMantiuk tmFattal makeGIMP}
: ${phase3Modules=adminModules delMediaFiles}
: ${PREVIEW_MODE=1}
: ${PREVIEW_SIZE=1024}
: ${DEL_FILES=dsc ca ais enf mantiuk fattal}
```

We start by running the modules of the first phase:

```
$ > hdrff phase1Modules
```

Except for the raw-conversion, the steps of this phase aren't very time-consuming. But now is the time to preview the images in `$TARGET_DIR/work` and discard (delete) all images and image-sequences which don't meet our quality standards. Also, we would edit the file `$TARGET_DIR/work/hdr-list.txt` to delete all the discarded images. Maybe we would also have to change the list itself, since the module findGroups uses a heuristic to find image-sequences, and this heuristic sometimes fails.

After this initial cleanup, which will save a lot of time later on, we run the second phase:

```
$ > hdrff phase2Modules
```

At this point, we are at the same point as after the simple workflow above. We could examine all generated files and discard a few more images and or image-sequences.

Running the modules of the third phase

```
$ > hdrff phase3Modules
```

will do the following:

- Delete intermediate files from the work-directory.
- Set the file-date of all generated files to the file-date of the respective original file.
- Move all files (original images and generated files) to the archive directory. The root of the archive-directory defaults to `$TARGET_DIR/archive`. The module mv2archive creates subdirectories below this root. These subdirectories are named in the format *yyyymmdd*, and every file will be moved to the date-directory corresponding to the date the image was taken.
- As a last step, the original files are deleted from the memory card.

It's a good idea to backup your archive-directory now. Paranoid people will only delete the original files from the memory card *after* having backed up the archive-directory.

6. A scenario with limited disk space

The full hdrff-workflow creates a lot of large intermediate tif-files. If you are low on disk space, you can employ one of the following strategies.

Strategy one will delete intermediate files as soon as possible:

```
$ > hdrff copyFiles name2lc fixDate setRO jpg2tif raw2tif findGroups
fixCA
$ > DEL_FILES="dsc_" hdrff delInterFiles
$ > hdrff alignImages
$ > DEL_FILES="ca_" hdrff delInterFiles
$ > hdrff enfuseImages makeHDR
$ > DEL_FILES="ais_" hdrff delInterFiles
$ > hdrff tmMantiuk tmFattal makeGIMP
$ > DEL_FILES="fattal_ mantiuk_ enf_" delInterFiles
$ > hdrff setOrigDate mv2Archive
```

This example assumes that you are only interested in the final GIMP-file. You can change the last *delInterFiles*-command if you want to keep additional files.

The second strategy processes only a subset of the images at a time and deletes the intermediate files at the end of each batch of files. The following script fragment will assumes this configuration:

```
: ${MEDIA:=/mnt/media}
: ${CP_LIMIT:=36}
: ${TARGET_DIR:=/data/images}
: ${IMG_EXT:=nef}
: ${IMG_PREFIX:=dsc_}
: ${IMG_DEPTH:=16}
: ${IMG_FIXCA:=1}
: ${IMG_ALIGN:=1}
: ${MODULES=enfuseModules makeHDR tmMantiuk tmFattal makeGIMP}
: ${DEL_FILES=dsc ca ais enf mantiuk fattal}
```

The added configuration variable `CP_LIMIT=36` limits each run to a batch of 36 images (of course you should choose a number small enough for the available disk-space).

```
while true; do
  hdrff
  [ $? -eq 12 ] && break
  hdrff adminModules delMediaFiles
done
```

The first module copyFiles only copies a maximum of `CP_LIMIT` images at a time. If there are no more images to copy, the module returns with a return-code of 12. The last module delMediaFiles also respects the value of `CP_LIMIT` and only deletes the images

already processed.

This approach also works if the first variant still produces too many intermediate files to fit on disk. But it has one drawback. If a sequence spans two batches, it wont be recognized correctly. You should examine the images at the end of one batch and the beginning of the next and if you find an interrupted sequence, you should copy the respective files back to `IMG_DIR`, crate the hdr-list.txt-file and start with the modules of phase 2 as described in the section above.

7. Writing your own modules

You can extend hdrff with your own modules. Hdrff will automatically load the files `$HOME/.hdrff/*.inc`. A module *foo* must implement as a minimum three functions *foo_help()*, *foo_check()* and *foo()*. In fact, you should also implement the function *foo_single()* and *foo_multi()*. These functions support parallel processing and image-filtering.

The best way to start your module is to create a template from an existing module:

```
$ > sed -e "s/noop/foo/g" /usr/local/lib/hdrff/noop.inc >
$HOME/.hdrff/foo.inc
```

Hdrff will already be happy and will load and execute the module (try `hdrff -H` or `hdrff foo`).

Now you have to edit the header and change the help-text. If your module needs some special external commands, you should also edit *foo_check()* and test for the existence. Have a look at the existing modules if you are unsure how to do it. Finally, you would implement the the function *foo*. Again, you could use existing modules as a template.

Usually, *foo()* only calls one of the functions *processListPar* (preferred) or *processListSeq*. The first will call multiple instances of *foo_single()* and *foo_multi* in parallel, the latter will sequentially process the hdr-list.txt-file.

The core module code goes into the functions *foo_single()* and/or *foo_multi()*. If your module only processes real image sequences, you would only implement the multi-variant (leaving the single-variant with the empty body of the template). Similarely, if you only process single images you only need the single-variant (see the module raw2seq for an example). Sometimes you have to implement both variants.

If you think your module is of general interest, post it in the open discussion forum of hdrff. I will be happy to add it to the hdrff-distribution with the next release.

8. Using hdrff4img

Since version 1.1, the hdrff-distribution contains an additional script: hdrff4img. The idea of this script is to provide support for hdr-tasks without the need to set up the whole

environment needed for hdrff. You would typically use `hdrff4img` for adhoc conversions of images. It is especially useful for testing parameters.

Let's start with an example. You have a number of raw-files and want to create a HDR-file. After that, you want to use various tone-mapping algorithms to process this HDR-file. You would use the following commands to for these tasks:

```
$ > hdrff4img -t "hdr" dsc_1234.nef dsc_1235.nef dsc_1236.nef
$ > hdrff4img -t "fattal"  hdr_1234.hdr
$ > hdrff4img -t "mantiuk"  hdr_1234.hdr
```

The first command creates the file `hdr_1234.hdr` in the current working-directory. After that, the following commands process this file to create `fattal_1234.tif` and `mantiuk_1234.tif`.

As you can see, with `hdrff4img` you can only do one thing at a time. It will take the input-files, guess the necessary modules needed for the given task, run these modules, delete intermediate files and only keep the target-file.

The `hdrff4img`-script expects you to pass the images as arguments and the *target-task* with the `-t` option. An additional option `-D` lets you specify the target-directory, which defaults to the current working directory. Passsing "+" to the `-D`-option will use the directory of the first argument image as the target-directory.

The *target-task* is not a module-name, but a name for a processing-level in the workflow. E.g. a valid value for target-task is "*align*. If you pass this target-task to `hdrff4img`, it will process the argument files and align them (it will also automatically identify image-sequences). Other valid values for target-task are *enfuse* to enfuse the images or *hdr* to create HDRs and so on. A list of valid values is available in the man-page or with `hdrff4img -h`.

Bear in mind that `hdrff4img` is not as flexible as `hdrff`. In order to be simple to use, it uses a number of heuristics to set hdrff-config-variables. It will also examine the argument images and guess their current processing-level to minimize the modules needed to achieve the target processing-level.

You should also pass only one type of file, e.g. either raw, jpg, tif or hdr. If you pass tif-files, they should have the same processing-level, e.g. all should have already CAs removed. And all files should have the same image-prefix.

Consider three tif-files that were manually converted from raw. You want to align these images and create a hdr-file. The necessary commands are:

```
$ > hdrff4img -t "hdr" img_1234.tif img_1235.tif img_1237.tif
```

This command will create the file `hdr_1234.hdr` in the current working directory. To process this file and create a number of different tone-mapped versions, you could run the commands as in the first example. If you want to also enfuse the files, you would use the

target-task *"enfuse"*. This will also realign the images, since all intermediate images were deleted. So it is more efficient to split up the processing:

```
$ > hdrff4img -t "align"  img_1234.tif img_1235.tif img_1237.tif
$ > hdrff4img -t "hdr"    ais_1234.tif ais_1235.tif ais_1237.tif
$ > hdrff4img -t "enfuse" ais_1234.tif ais_1235.tif ais_1237.tif
```

The first command will align the images and create the files with the ais-prefix which are then used by the following two commands. As an alternative, you could pass the `-k`-option to `hdrff4img`. With this option the script will keep *all* intermediate files.

`hdrff4img` has two options to explicitly group images, thus overriding the automatic sequence-detection (which sometimes failes). If you use the `-g`-option, you declare that all images belong to a single image-sequence. Using the `-G`-option, you explicitly pass the image-groups, e.g.

```
$ > hdrff4img -t "hdr" -G "1-3 4-6 7-11" *.nef
```

Here you would define three groups, the first two with three images each, the last group has five members.

If you want to change the default behaviour, you can either pass a standard hdrff-configuration-file with the `-C`-option, or set the configuration-variables on the commandline.