A powerful renaming utility

Features

- Rename multiple files and/or directories at once (batch rename).
- Automatically create parent directories for renamed files.
- Use regex capture groups to extract parts of the input file name to be used in the output filename.
- Use the power of jinja2 templates to make complex renaming operations simple
  - Use filters to increment enumerations, add padding, switch between camel and snake case, etc.
- Easily increment an integer value in the filename of a set of files. i.e. (rename file-1.txt to file-2.txt)
- *All* actions are analyzed to detect errors (missing inputs, output name collisions, overwriting files) *before* making any changes. If *any* error is detected, no actions are executed.
  - Multiple input files being renamed to the same output file is an *error* unless the output is a directory.
  - Files that would be renamed to an existing file that is not one of the input files to be renamed is an *error* unless the –overwrite option is given.
  - Input files that do not exist is an *error*.
  - Input files that appear multiple times is an ___error_.
- Match full path, just the name, or a portion of either.

Motivation

Why another batch renaming tool?

PowerMV has goals similar to

- rnr
- nomino
- brename
- rename
- rnm

Of these utilities, I have used `rename` the most, and recently started using `rnr`. Both tools are nice and work for 99% of my use cases. However, there is one specfic use case that I occationally have when working with files created by/for some physics simulation or demos for Phys 312 class, and that is batch renaming with increment/decrement of integers in the filename. For example, say I have some demo files:

```
$ ls
01-text_files.sh
02-text_editors.sh
```

```
03-file_redirection.sh
```

I have these files named so that they will be loaded (by my pygsc utility) in order. Now say I want to add a demo at the beginning of the tutorial for some preliminary stuff. I create a filed named `01-preliminaries.sh`. But before I do, I would like to rename all of the existing scripts to increment their index:

```
01-text_files.sh       -> 02-text_files.sh
02-text_editors.sh     -> 03-text_editors.sh
03-file_redirection.sh -> 04-file_redirection.sh
```

I would like to have way to do this rename automatically. There are some tools (like ranger) that allow you to do batch renaming and edit the file rename operations in a text file, so you can use vim's ctl-a and ctl-x to help do the rename quickly. However, there are some situations that you need to be careful with.

Let say I have a set of enumerated input configuration files.

```
$ ls
config-01.yml
config-02.yml
config-03.yml
config-04.yml
config-05.yml
```

If I want to rename these to

```
$ ls
config-02.yml
config-03.yml
config-04.yml
config-05.yml
config-06.yml
```

there is a possibility that I will accidentally delete files. If `config-01.yml` gets renamed to `config-02.yml` *first*, then when `config-02.yml` is renamed to `config-03.yml`, it will actually be a copy of the original `config-01.yml`. If all operations go in order, you will end up with one file.

```
$ ls
config-06.yml
```

where the contents of `config-06.yml` will be the contents of the original `config-01.yml`. Clearly not what was intended.

PowerMV aims to address these problems and make file renaming with incremented/decremented enumeration indecies possible and easy.

Install

You can install PowerMV with `pip`, `pipx`, `uv`, or your favorite Python package manger.

```
$ pip install powermv
```

```
$ pipx install powermv
```

```
$ uv tool install powermv
```

Usage

```
$ powermv --help | tr -cd '\11\12\15\40-\176'
Usage: powermv COMMAND [ARGS] [OPTIONS]

Batch move files with the power of jinja2 templates.

With great power comes great responsibility...

 Arguments
 *  MATCH_PATTERN     Pattern to match input filenames against. [required]
 *  REPLACE_TEMPLATE  Jinja2 template to render output filename with.
                      [required]
 *  FILES             [required]

 Commands
 inc         Increment integer enumerations in filenames. This is a shorthand
             for
 --help -h   Display this message and exit.
 --version   Display application version.

 Parameters
 EXECUTE --execute          -x  Execute move operations (by default, nothing
   --no-execute                  is moved, only a dry-run is performed).
                                 [default: False]
 NAME-ONLY --name-only      -n  [default: False]
   --no-name-only
 OVERWRITE --overwrite          Proceed with executing operations even if they
   --no-overwrite               would overwrite existing files. [default:
                                 False]
 VERBOSE --verbose          -v  Print extra status information. [default:
   --no-verbose                  False]
 QUIET --quiet --no-quiet   -q  Don't print status information. [default:
                                 False]
```

Examples

Rename a series of files that are enumerated, incrementing the enumeration by one.

The original motivation for PowerMV...

```
$ echo 1 > file-1.txt
$ echo 2 > file-2.txt
$ echo 3 > file-3.txt
$ ls
file-1.txt
file-2.txt
file-3.txt
$ powermv 'file-(\d).txt' 'file-{{_1|inc}}.txt' *
Ready to perform move operations
file-3.txt -> file-4.txt
file-2.txt -> file-3.txt
file-1.txt -> file-2.txt
$ powermv 'file-(\d).txt' 'file-{{_1|inc}}.txt' * -x
Ready to perform move operations
file-3.txt -> file-4.txt
file-2.txt -> file-3.txt
file-1.txt -> file-2.txt
$ ls
file-2.txt
file-3.txt
file-4.txt
$ cat file-2.txt
1
```

A couple of things to note. First, PowerMV does not do anything by default. All move operations are created, analyzed, ordered, and displayed, but nothing happens. If you want to execute the move operations, you give the `-x` option (alias for `--execute`). Second, note how PowerMV has ordered the move operations so that `file-3.txt` gets moved *before* `file-2.txt` get moved to `file-3.txt`. If PowerMV detects that a file will be renamed to a file that is also going to be renamed, it will make sure that latter happens first.

PowerMV also provides an `inc` command as a shorthand for this

```
$ echo 1 > file-1.txt
$ echo 2 > file-2.txt
$ echo 3 > file-3.txt
$ ls
file-1.txt
file-2.txt
file-3.txt
```

```
$ powermv inc *
Ready to perform move operations
file-3.txt -> file-4.txt
file-2.txt -> file-3.txt
file-1.txt -> file-2.txt
$ powermv inc * -c 2
Ready to perform move operations
file-2.txt -> file-4.txt
file-3.txt -> file-5.txt
file-1.txt -> file-3.txt
$ powermv inc * -c 2 -p 2
Ready to perform move operations
file-1.txt -> file-03.txt
file-2.txt -> file-04.txt
file-3.txt -> file-05.txt
```

Rename enumerated files, incrementing enumeration by two.

```
$ touch file-1.txt file-2.txt
$ powermv '(\d)' '{{_1|inc(2)}}' * -x
Ready to perform move operations
file-1.txt -> file-3.txt
file-2.txt -> file-4.txt
```

Rename enumerated files, decrementing enumeration by one.

```
$ echo 1 > file-1.txt
$ echo 2 > file-2.txt
$ echo 3 > file-3.txt
$ ls
file-1.txt
file-2.txt
file-3.txt
$ powermv 'file-(\d).txt' 'file-{{_1|inc}}.txt' *
Ready to perform move operations
file-3.txt -> file-4.txt
file-2.txt -> file-3.txt
file-1.txt -> file-2.txt
$ powermv 'file-(\d).txt' 'file-{{_1|inc}}.txt' * -x
Ready to perform move operations
file-3.txt -> file-4.txt
file-2.txt -> file-3.txt
file-1.txt -> file-2.txt
$ ls
file-2.txt
file-3.txt
```

```
file-4.txt
$ cat file-2.txt
1
```

Rename enumerated files to increase the padding used in the enumeration.

```
$ echo 1 > file-1.txt
$ echo 2 > file-2.txt
$ echo 3 > file-3.txt
$ powermv 'file-(\d).txt' 'data_file-{{_1|pad(2)}}.txt' * -x
Ready to perform move operations
file-1.txt -> data_file-01.txt
file-2.txt -> data_file-02.txt
file-3.txt -> data_file-03.txt
$ ls
data_file-01.txt
data_file-02.txt
data_file-03.txt
```

Move files into their own directories.

```
$ echo 1 > file-1.txt
$ echo 2 > file-2.txt
$ echo 3 > file-3.txt
$ powermv 'file-(\d).txt' 'dir-{{_1}}/file.txt' * -x
Ready to perform move operations
file-1.txt -> dir-1/file.txt
file-2.txt -> dir-2/file.txt
file-3.txt -> dir-3/file.txt
$ ls
dir-1
dir-2
dir-3
$ head */*
==> dir-1/file.txt <==
1

==> dir-2/file.txt <==
2

==> dir-3/file.txt <==
3
```

Match files to move using a part of the full path, a part of the file name, or the full filename.

```
$ mkdir -p dir/level1/level2/
$ touch dir/level1/level2/file-1.txt
$ touch dir/level1/level2/file-2.txt
$ touch dir/level1/level2/datafile-1.txt
$ touch dir/level1/level2/datafile-2.txt
$ powermv '(\d)' '{{_1|inc}}' dir/*/*/*
Ready to perform move operations
dir/level1/level2/datafile-1.txt -> dir/level2/level2/datafile-1.txt
dir/level1/level2/datafile-2.txt -> dir/level2/level2/datafile-2.txt
dir/level1/level2/file-1.txt -> dir/level2/level2/file-1.txt
dir/level1/level2/file-2.txt -> dir/level2/level2/file-2.txt
$ powermv '(\d)' '{{_1|inc}}' dir/*/*/* -n
Ready to perform move operations
dir/level1/level2/datafile-2.txt -> dir/level1/level2/datafile-3.txt
dir/level1/level2/file-2.txt -> dir/level1/level2/file-3.txt
dir/level1/level2/datafile-1.txt -> dir/level1/level2/datafile-2.txt
dir/level1/level2/file-1.txt -> dir/level1/level2/file-2.txt
$ powermv 'file-(\d)\.txt' 'FILE-{{_1|inc}}.txt' dir/*/*/* -n
Ready to perform move operations
dir/level1/level2/datafile-1.txt -> dir/level1/level2/dataFILE-2.txt
dir/level1/level2/datafile-2.txt -> dir/level1/level2/dataFILE-3.txt
dir/level1/level2/file-1.txt -> dir/level1/level2/FILE-2.txt
dir/level1/level2/file-2.txt -> dir/level1/level2/FILE-3.txt
$ powermv '^file-(\d)\.txt' 'FILE-{{_1|inc}}.txt' dir/*/*/* -n
Ready to perform move operations
dir/level1/level2/file-1.txt -> dir/level1/level2/FILE-2.txt
dir/level1/level2/file-2.txt -> dir/level1/level2/FILE-3.txt
```

Switch between camel case, snake case, and space case.

```
$ touch "one_two_three.txt" "four five.txt"
$ powermv "(.*)\.txt" "{{_1|CamelCase}}.txt" * -x
Ready to perform move operations
four five.txt -> FourFive.txt
one_two_three.txt -> OneTwoThree.txt
$ ls
FourFive.txt
OneTwoThree.txt
$ powermv "(.*)\.txt" "{{_1|snake_case}}.txt" * -x
Ready to perform move operations
FourFive.txt -> four_five.txt
OneTwoThree.txt -> one_two_three.txt
$ ls
```

```
four_five.txt
one_two_three.txt
$ powermv "(.*)\.txt" "{{_1|space_case}}.txt" * -x
Ready to perform move operations
four_five.txt -> four five.txt
one_two_three.txt -> one two three.txt
$ ls
four five.txt
one two three.txt
```

How it works

PowerMV builds a set of "move operations" that need to be executed. Each move operation consists of an "input" (a file/directory that exists and should be renamed/moved) and an "output" (a file/directory that the input will be moved to). The move operation set is built using a "match pattern", a "replace template", and a list of files. All are passed as arguments.

Only files that match the *match pattern* will be renamed, so files that should not be renamed can be passed as arguments. This is useful because you can just use a `*` to pass all files in the current directory, and only the files matching the *match pattern* are added to the move operation set (just like the `rename` command).

To build the move operation set, PowerMV check that a file matches the *match pattern*. If it does match, then the *replacement text* is rendered using the *replacement template* (a Jinja2 template). PowerMV automatically creates a context for the *replacement template* from the *match pattern*. Named capture groups are injected into the template as variables with the capture group name. Unnamed capture groups are injected into the template as variables named by the capture group index. The first unnamed capture group will be named `_1`, the second `_2`, etc.

Capture groups are converted to integers if possible. This means they will be injected into the Jinja2 template as integers and you can use them in simple math expressions. A special capture group named `_0` contains the entire match as a string.

If a file is given more than once, it is skipped after the first instance. This means that `* *` as an argument will just work.

Once the move operation set is built, it is analyzed for errors. If the input for a move operation does not exist, it will be reported as an error. If the same input appears multiple times in the move operation set, it will be reported as an error. If two move operations have the same output, it is reported as an error *unless* the output is a directory. Outputs that are detected as directories that already exist are automatically treated as directories. If the output does not exist, but should be created during the rename, you can indicate that it should be treated

as a directory by adding a `/` to the end of the replace template.

If any outputs would overwrite an existing file that is *not* an input to another move operation, it is reported as an error. This protects from accidentally deleting a file by giving an incorrect replace template. If you want the oeverwrite to occur you can pass the `--overwrite` option to ignore the error.

If no errors are detected, then (and only then) PowerMV executes each move operation.