A powerful renaming utility

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

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' *
Building move operations set
Analyzing move operations set
Ordering move operations
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
```

```
Building move operations set
Analyzing move operations set
Ordering move operations
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.

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
Building move operations set
Analyzing move operations set
Ordering move operations
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
```

```
Building move operations set
Analyzing move operations set
Ordering move operations
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
```

More examples to come... (maybe)