Renumbering files using prename, the Perl script for renaming files

Ronald A. Olsson
Department of Computer Science
University of California, Davis
Davis, CA, 95616 USA

olsson@cs.ucdavis.edu

March 29, 2016

Summary

I'll show how to use the Perl prename script, available on many Linux systems, to solve common file renumbering problems. I'll also extend prename slightly so it can solve additional such problems. Previous solutions to renumbering don't provide as much flexibility as using (modified) prename.

Motivation

I often find that I have a bunch of files that I want to renumber. For example, I might have test files named tXX, where XX represents some digits and I decide to add a new test file between t15 and t16. So I need to rename test files t16, t17, etc. Or I might want to delete a test file and rename the files so I don't have any gaps. As another example, I might decide to rearrange my lecture notes by moving lecture05 to lecture10 and renaming existing lecture06 through lecture10 to lecture05 through lecture09.

A quick Google search shows that these kinds of renumbering problems are fairly common, and quite a few solutions have been proposed, such as RESOURCE[1], RESOURCE[2], RESOURCE[3], RESOURCE[4], RESOURCE[5]. RESOURCE[6], RESOURCE[7], and RESOURCE[8].

These solutions include several nice shell scripts, a custom Perl script, and one that uses Perl's prename tool. However, for my purposes, I found none of these solutions lets me do all the kinds of renumbering I need to do. I was tempted to write my own script (which would be fun, but likely time consuming), but after a bit of exploration I discovered that I could use Perl's rename script (aka prename) to solve some of my renumbering problems, and then by making minor tweaks to Perl's rename script (available as RESOURCE[9]) I could solve my remaining renumbering problems.

Linux rename commands

I started by considering the Linux rename commands, which are useful for renaming many files in a single command. On my local Linux systems, I find two rename commands, which I'll call rename.ul and prename. rename.ul is from the util-linux package RESOURCE[10]. prename is a short Perl script. On my local Ubuntu systems, /usr/bin/rename resolves to prename. On my local Fedora systems, /usr/bin/rename resolves to rename.ul. (For more on the various renames, see RESOURCE[11].)

Here's a simple example to illustrate the utility of the two rename commands. We have a bunch of files named for Joe Bob whose names need to be upgraded to Joe Robert. To be concrete, suppose we have the specific ”source” filenames shown in the ”before” and we want to rename them to the corresponding ”target” filenames shown in the ”after”.

before:
joeBob.c joeBob.h joeBob.o joeBob.out

after:
joeRobert.c joeRobert.h joeRobert.o joeRobert.out

I can use a rename.ul command that specifies that the expression Bob is to be replaced by Robert in each filename:

```
rename.ul Bob Robert joeBob*
```

Or I can use a prename command with a simple Perl substitution (s///) that’s performed on each filename:

```
prename 's/Bob/Robert/' joeBob*
```

The prename command's use of a Perl expression is elegant and, I'll show later, provides nice flexibility for our purposes of renumbering. To use it, though, you do need to know how to write at least simple Perl expressions.

Kinds of renumbering

I break renumbering into three different kinds: shifting, compacting, and reversing. In the following, I'll explain these different kinds of renumbering and give examples to illustrate each.

Shifting up

Let's start with a few files whose names we want to shift up by 10

before:
t11 t12 t15 t16 t19

after:
t21 t22 t25 t26 t29

```
prename 's/(\d+)/$1+10/e' t*
```

Here, I'm using Perl's s///e operator. It's similar to the s/// I used earlier, but its replacement text will be the value of a Perl expression, which can reference the matched text (such as `$1`) and can invoke other operators and functions. So, the above expression says the replacement text is the result of adding 10 to the matched digits. (This solution is based on the one given in RESOURCE[4].)

This script works for this specific problem, but as we'll see later it might not work in all cases and we'll see then how to fix it.

Shifting down

Here I want to shift a bunch of filenames down by 6:

before:
t11 t12 t15 t16 t19

after:
t05 t06 t09 t10 t13

```
prename 's/(\d+)/sprintf("%02d",$1-6)/e' t*
```

This prename command is similar to the previous one, but since I still want file names to have two digits (for example, so they sort nicely when I use ls), I use Perl's sprintf to output the new file numbers. More later on the number of digits in filenames.

This example and the previous, shifting up example illustrate how prename is more flexible than rename.ul. I can easily use rename.ul for the shifting up example since we're shifting by 10:

```
rename.ul t1 t2 t*
```

This rename.ul command specifies that the expression t1 is to be replaced by t2 in each filename.

But, shifting down (or up) by 6 doesn't fit into the pure textual style of replacements in rename.ul.

Compacting up

I want to eliminate any gaps within the file numbers.

before:
t11 t12 t15 t16 t19

after:
t11 t12 t13 t14 t15

I call this direction "up" since we start with the lowest file number and work upward.

Something like one of the prename expressions I've used before won't work here. The problem is that the expression operates on one filename at a time and keeps no state information, whereas here we need to know the number of the previous file in order to renumber each file.

To give me what I want, I tweaked prename slightly by adding a variable `$renameCounter`. This variable will be available to the user within the replacement expression, as we'll see in the upcoming examples. The value of `$renameCounter` is the number of filenames processed by prename so far. The code changes to the prename script are simple: declare the `$renameCounter` variable, initialize it, and increment it within prename's loop. RESOURCE[9] has the code for the original prename and my version, myPrenameV1. (This is version 1 of myPrename; later I need to make one more tweak, after which I'll call my script just plain myPrename.)

Using myPrenameV1, compacting up is straightforward:

```
myPrenameV1 \
  's/(\d+)/sprintf("%02d",11+$renameCounter)/e' \
  t*
```

Compacting down

Compacting down renames file from right to left

> before:
> t11 t12 t15 t16 t19
>
> after:
> t15 t16 t17 t18 t19

The expression for compacting down is similar to that we saw for compacting up, but we need to process the files in right-to-left order, so I use the output of `ls -r` instead of `*` to get the filenames in reverse order.

```
myPrenameV1 \
  's/(\d+)/sprintf("%02d",19-$renameCounter)/e' \
  `ls -r t*`
```

Reversing

First, consider the simple case where the sequence of filenames has no gaps such as:

> before:
> t11 t12 t13 t14 t15
>
> after:
> t15 t14 t13 t12 t11

Conceptually, this is simple: just swap corresponding pairs of files. However, prename deals with only one file at a time, so we can't swap. And we can't rename just one of a pair of files (say t11) to where it should end up (t15) since that would overwrite the second file. Instead, we'll rename the source files to names close to the target names and then later rename the files to the desired names.

```
prename 's/(\d+)/26-$1."tmp"/e' t*
```

In this prename command, the magic number 26 comes from 15+11, so the numbering works out as it needs to. I use Perl's concatentation operator "." to append "tmp" to the target names. For example, t11 is renamed to t15tmp. After they've all been renamed, I then use simple substitution to delete the "tmp" in the final rename:

```
prename 's/tmp//' t*tmp
```

Reversing with gaps, maintaining "distance"

Here the sequence of filenames has some gaps such as in:

before:
t11 t12 t15 t16 t19

after:
t19 t18 t15 t14 t11

Those gaps are maintained by having the file numbers keep the same distance from others. I mean, for example, that in the "before" the names t12 and t15 differed by 3 and in "after" their new names t18 and t15 still differ by 3 (actually now by -3).

```
prename 's/(\d+)/30-$1."tmp"/e' t*
```

In this prename command, I again used the "tmp" files and the magic number 30 comes from 19+11. I then use the same prename as before to delete the "tmp".

Reversing with gaps, maintaining original filenames

Here we just swap the first and last files, the second and second-to-last files, etc.

before:
t11 t12 t15 t16 t19

after:
t19 t16 t15 t12 t11

My plan is to access the names of the files by using the `$ARGV` variable (which contains the command line arguments) within the prename script. I'll use

```
$ARGV[$#ARGV-$renameCounter]
```

to get the target names (`$#ARGV` gives the number of elements in `$ARGV`). I admit that my first attempt

```
myPrenameV1 \
  's/t(\d+)/$ARGV[$#ARGV-$renameCounter]."tmp"/e' t*
```

was wrong and ended up with odd-looking filenames.

before:
t11 t12 t15 t16 t19

after:
t19tmp t16tmp t15tmp t16tmptmp t19tmptmp

I forgot that prename is directly modifying `ARGV`, so it works fine for the first three filenames, but for the fourth (and similarly for the fifth) it's using the already modified `ARGV[2]`, whose value was already set to `t16tmp`!

I fixed that problem by making another simple tweak to prename, so that it first saves a copy of `ARGV`, which I named `copiedARGV`, and then the prename code makes changes to that copy instead of `ARGV`. As this is my last tweak to prename, I named this version just myPrename. Using this new version, my formerly broken code now works.

```
myPrename \
  's/t(\d+)/$ARGV[$#ARGV-$renameCounter]."tmp"/e' t*
```

I then use the same prename as before to delete the "tmp" from the file names.

Collisions

To keep things simple, I've assumed in the examples that the files listed are the only ones that exist in the directory where I'm doing the renumbering. That's often not the case. For example, in the shifting up example, the directory might already have other files that we don't want to move. So, we'll need to be more specific in saying which filenames to shift such as

```
prename 's/(\d+)/$1+10/e' t1[1-9]
```

But, even then, the directory might already contain another file whose name is a target of the renumbering, say a file named "t22". I call this kind of situation a collision. Here's the result of the prename command

```
before:
t11 t12 t15 t16 t19 t22

after:
t21 t12 t25 t26 t29 t22
```

Note how it did not rename "t12" and left the existing "t22" file intact;

prename reports that as a warning. If you actually do want to overwrite that file, prename's `--force` option will do it.

It's often handy to see what prename would do without actually having it do the renaming. prename's `--no-act` option lists the sequence of renames it would perform.

It is important to note that my shifting up code given earlier might encounter collisions. For example, suppose we want to shift up by just 1,

```
prename 's/(\d+)/$1+1/e' t*
```

Then prename will detect collisions, and report warnings, for "t11" and "t15".

> before:
> t11 t12 t15 t16 t19
>
> after:
> t11 t13 t15 t17 t20

To avoid such collisions, I'll change the command to process the filenames in reverse order, as we've seen before. That way, the target in the original collision ("t16" or "t12"). will already have been renamed when it's time to rename the source ("t15" or "t11").

```
prename 's/(\d+)/$1+1/e' `ls -r t*`
```

before:
t11 t12 t15 t16 t19

after:
t12 t13 t16 t17 t20

So, in a sense these avoidable collisions are "false" collisions. Similarly, shifting down should be done in normal, left-to-right order, as we did earlier.

Overflow and underflow

We've seen in the shifting down example the possibility of underflow. The source files had two digits, but some of the target files had only one digit.

Recall that our solution was to use `sprintf` to maintain two digits, although if it's fine with you to have just one digit there, you can drop the `sprintf` from the code. Similarly, overflow can occur as in

```
prename 's/(\d+)/$1+83/e' `ls -r t*`
```

before:
t11 t12 t15 t16 t19

after:
t94 t95 t98 t99 t102

Again, that might be acceptable or perhaps the files with two digits in their names should be renamed with a leading zero That's easily done by using `sprintf` in the code or by using a separate prename command afterward.

Another, likely more serious, sense of underflow occurs with this script

```
prename 's/(\d+)/sprintf("%02d",$1-15)/e' t*
```

in this scenario

before:
t11 t12 t15 t16 t19

after:
t-4 t-3 t00 t01 t04

Those renumbered file names are not likely what's wanted.

Two more complicated scenarios

Suppose we have a bunch of files with .e and .ff suffixes whose names are within the ranges t01-t17, t21-t26, and t31-t39. We want to compact them so they're numbered t01, t02, etc. We can use two myPrename commands, one for each suffix, similar to what we've seen earlier.

```
myPrename \
  's/(\d+)/sprintf("%02d",1+$renameCounter)/e' \
  t[0-9][0-9].e
```

```
myPrename \
  's/(\d+)/sprintf("%02d",1+$renameCounter)/e' \
  t[0-9][0-9].ff
```

Note how we use shell's filename notation, such as `t[0-9][0-9].e`, here to limit the files we rename. We could use just `t*.e` here too, but we're trying to be more specific. So, this command will not unintentionally renumber a file named something like `test4fun.e`.

Alternatively to using two prename commands, we could write a single command to do the same job.

```
myPrename \
's:(\d+):'\
'sprintf("%02d",1+int($renameCounter/2)):e' \
  t[0-9][0-9].@(e|ff)
```

First I changed the delimiter on s///e from the usual "/" to ":" since I want to do division (using "/") within the replacement expression. We need to divide by 2, since we're renumbering two kinds of files. I also used `int` to cast (truncate) the floating point result of the division to an integer.

The specification of the files uses bash's extended pattern matching (`extglob` needs to be enabled via `shopt -s extglob`). Using it will present the filenames to prename in the order t01.e, t01.ff, t02.e, t02.ff, and so on so as to get the right renumbering. If I had instead used just `t[0-9][0-9].e t[0-9][0-9].ff`, that would first present all the `.e` files and then all the `.ff` files.

As a second example, in writing a book, I might have filenames of the form chapXXsecYY. I decide that section 3 in chapter 3 would make more sense as the last section (section 8) of that chapter. So I need to move present sec03 to be chap03's last section

```
mv chap03sec03 chap03sec09
```

and then renumber the other sections and the new chap03sec09 so there's no gap.

```
prename 's/(sec)(\d+)/$1.sprintf("%02d",$2-1)/e' \
        chap03sec0[3-9]
```

The expression here specifies to match ”sec” followed by a number; if we used our earlier shift down code, its expression would match the chapter number. The replacement uses `$1`, whose value is ”sec”, and `$2`, whose value is the matched number. The filenames given to prename specify just those section we want shifted down.

Closing thoughts

I’ve used relatively simple Perl expressions. If necessary, the Perl expressions can be much more complicated as seen for example in RESOURCE[12].

I’ve shown the examples for renumbering filenames where the sequence numbers are decimal numbers. You can also define renumbering commands for when sequence ”numbers” are character sequences or octal or hexadecimal numbers by using slightly different Perl expressions such as using `[a-z]` to match lowercase characters.

If you use these commands often and don’t want to re-type the Perl expressions all the time, you might find it handy to set up aliases or bash functions named, say, shiftup, shiftdown, compactup, compactdown, and so on.

Finally, two caveats:

1. Before doing any kind of renaming or renumbering of files, be sure to save a copy of the original in case you get the command wrong.
2. Changing around file names (especially swapping some of their names) might subvert backup systems or version control systems

Acknowledgements

Todd Williamson, Ali Minty, and Nancy Wilson provided helpful comments on this work and earlier drafts of this paper.

Resources

[1] Ben Okopnik. 2-cent Tip: Renumbering files. http://linuxgazette.net/178/misc/lg/2_cent_tip___renumbering_files.html.

[2] Renumber a file sequence. https://www.lumalab.net/index/trans/en/page/post/t/269/f/7/#0.

[3] Rename files by incrementing a number within the filename. http://unix.stackexchange.com/questions/40523/rename-files-by-incrementing-a-number-within-the-filename.

[4] Changing the sequence numbers of files in Linux. http://superuser.com/questions/636994/changing-the-sequence-numbers-of-files-in-linux.

[5] Dave Taylor. Work the shell - scripting common file rename operations. http://www.linuxjournal.com/article/10885.

[6] Dave Taylor. Can I sequentially rename files in Linux? http://www.askdavetaylor.com/sequentially_rename_files_linux_mac_script/.

[7] Dave Taylor. How can I bulk rename Apple iPhone screen captures? http://www.askdavetaylor.com/how_to_bulk_rename_apple_iphone_screen_captures/.

[8] Dave Taylor. How do I rename hundreds of files at once? http://www.askdavetaylor.com/how_do_i_rename_hundreds_of_files_at_once/.

[9] Ronald A. Olsson. myPrename. http://www.cs.ucdavis.edu/~olsson/research/myPrename.

[10] util-linux package. ftp://ftp.kernel.org/pub/linux/utils/util-linux/.

[11] What's with all the renames. http://unix.stackexchange.com/questions/229230/whats-with-all-the-renames.

[12] MMV mass file rename. http://unix.stackexchange.com/questions/174387/mmv-mass-file-rename.