Subject: the genius of Unix
From: Joseph Picone <joseph.picone@gmail.com>
Date: 6/6/23, 9:25 PM
To: nedc_research <nedc_research@googlegroups.com>
CC: ECE 1111 <temple_engineering_ece1111@googlegroups.com>

Unix came along at a time when operating systems were getting extremely complicated and specific. Commands had long argument lists with obscure options. DOS, if some of you remember that, was structured like this and really reflected an IBM mindset.

Unix was built with a minimalist mindset. Create simple, general purpose commands. Let users combine these commands using things like pipes to build much more specific capabilities.

That was 40+ years ago and the philosophy is still paying off.

There are tons of really useful commands to process text in Unix. The most basic are things like grep. More sophisticated commands including cut, sort, uniq. Another level up are general parsers like sed, lex and yacc.

If you want to see how far you can go with these things, take a look at this tutorial:

https://isip.piconepress.com/courses/temple/ece_1111/resources/tutorials/unix_for_poets/unix_for_poets.pdf

You can slice and dice files many different ways - by line, by column, etc. You can count lines, words, etc. using sort and uniq, and even do things like histograms. With sed, lex and yacc, almost anything is possible.

But the amazing thing is these tools were developed in the late 1970's, when computing was very primitive, and yet the developers had a vision that has held up over time. We see this minimalist approach being successful in many engineering disciplines. Keeping designs simple and flexible almost always pays off over complex designs. (One notable exception to this is a GPU chip, but you will have to take ECE 4822 to understand why.)

-Joe

=======
Subject: replicating directory structures in databases
Date: Sun, 4 Jun 2023 13:07:56 -0500
From: Joseph Picone <joseph.picone@gmail.com>
To: nedc_research <nedc_research@googlegroups.com>
CC: ECE 1111 <temple_engineering_ece1111@googlegroups.com>

One of the powerful things about the command line environment we support in ISIP code is the "replace directory" set of options. This makes it very easy to process data and replicate a portion of the source directory tree in the destination directory. For example, suppose your input is of the form:

```
/one/two/three/four/f001.edf
/one/two/three/five/f002.edf
/one/two/nine/four/f003.edf
...
```

The replace dir options on most of our tools allow you to run the input list and automatically create, or replicate the output directory structure:

```
nedc_command.py -r /one/two -o /home/output -e edfx files.list

/home/output/three/four/f001.edf
/home/output/three/five/f002.edf
/home/output/nine/four/f003.edf
...
```

But how do you do this when building a new database? For example, suppose your task is to copy

500 files from a database of 14,000 files? How do you do that in a way that easily replicates the source directory structure in the output?

Of course, emacs macros and a handy command "mkdir -p" make this easy:

(1) Generate a list of inputs:

```
/one/two/three/four/f001.edf
/one/two/three/five/f002.edf
/one/two/nine/four/f003.edf
```

(2) Use an emacs macro to convert this list file, which is a text file, to a script:

```
#!/bin/sh
mkdir -p /home/output/three/four
mkdir -p /home/output/three/five
mkdir -p /home/output/nine/four
```

(3) Then convert the original list to a script that copies:

```
#!/bin/sh
cp /one/two/three/four/f001.edf /home/output/three/four/
cp /one/two/three/five/f002.edf /home/output/three/five/
cp /one/two/nine/four/f003.edf /home/output/nine/four
```

"mkdir -p" will make all of the directory structure. You don't have to make it level by level manually. It will create the entire tree, or whatever pieces of it don't exist. You can run it multiple times with no damage. It will add to the existing tree or ignore parts which already exist.

Though there are other ways to do this, including using rsync with an exclusion list, the combination of emacs macros and basic Unix commands/scripting is a really powerful and easy to remember way to do this. When building databases like TUH EEG or TUH DPATH, which involve hundreds of thousands of files, these tools can be incredibly valuable.

Some of you might ask "teach me how to do this". But it is really hard to teach because we make it up as we go along. Once you master the philosophy behind emacs macros, and you understand enough about Unix to be dangerous, you can solve these kinds of complicated problems very easily by breaking them into a series of steps, and implementing each step with a script generated by a macro.

To get started, just write down the sequence of steps you have to go through to accomplish your task, and then think about how each step can be implemented with a series of Unix commands. Then turn those commands into scripts using emacs macros.

-Joe