Care and feeding of one-liners

Andy Adler

Those are my principles, and if you don't like them... well, I have others.

-- Groucho Marx
Problem

- You're a "cool perl dood"; you can whip up a one liner in 1 minute.
- But actually, after testing it, doing the work, and checking that it did it right, you've probably spent 10-20 minutes.
- At 2 one-liners per week, this is 15-30 hours per year.
- Often, you need the same, or very similar code again and again and again ...
Some ideas

1. Oh well, that’s life!
   - The virtues of lazyness

2. Keep a notebook
   - But that means using paper ... yuck

3. Save each one to a file
   - But then its not a one-liner
   - What order are they used?

4. Post them to perlmonks ...
My idea: Makefile

- Put the code into Makefiles, and check into CVS.

- Advantages:
  - Made for little bits of code
  - Easy to integrate bash, perl, octave, etc.
  - Keeps track of order of operations

- Disadvantages:
  - Different *make* programs
  - *make* uses ‘$’ character as special
Example: keep only some files

cleanresults:
find -type f -name ims\*jpg | perl -n
  -e 'chomp;'
  -e '($$n)=/ims\d-(\d+)/\.jpg$$/ or '
  -e 'die "name:$$_";'
  -e 'unless ($$n=~/(0\d|15|20|30)01)/) {'
    -e 'print "del $$_\n"'
    -e 'unlink $$_ '
    -e 'or die "Cant unlink $$_;"'
  -e'}'
Example: test file differences

cleanresults same:
for dir in *_results/test? ; do \
  echo "Cleaning dir=\$dir" ; \
pfn="" ; \
  ls -r \$dir/ims*.jpg | \ 
  while read fn ; do \ 
    if [ -z \$pfn ] ; then \ 
      pfn=\$fn ; \ 
    else \ 
      if [ -z "\`diff -q \$fn \$pfn`" ] ; then\ 
        rm \$pfn ; \ 
      else \ 
        echo "\$pfn is different" ; \ 
      fi ; \ 
      pfn=\$fn ; \ 
    fi ; \ 
  done ; \
done
Example: create graphs

```bash
ERRGRAPH= dir1/test1.gif \  
dir2/test2.gif

$(ERRGRAPH): %.gif: % .txt.bz2
  ( echo "dd=[...]; \n    bzip2 -dc $< ; \n    echo "];"; \n    echo " \n    dd(1:420:length(dd),:)=[]; \n    eopen(`pwd`/@.ps'); \n    eglobpar; \n    ePlotAreaWidth=50; \n    ePlotAreaHeight=60; \n    eplot(dd(:,1),dd(:,2),',',0); \n    eclose; "; \n  ) | octave -q
convert -crop 0x0 $@.ps $@
```

List of files to operate on
Rule to create gif from txt.bz2
Insert entire file into input stream
Octave code to create graph in *ps file
imagemagick to convert to gif
Example: create graphs

```perl
results.html: $(ERRGRAPH)
perl -MFile::Find -w\n  -e 'print q{
  -e '<HTML><HEAD><TITLE>$@</TITLE></HEAD>''
  -e '<BODY><H1>$@</H1>''
  -e '<H3>Target Image</H3>''
  -e '<IMG SRC="./imt1-targ.jpg">''
  -e '};''
  -e 'for my $dir qw($^) {'
    -e 'my @files=();'
    -e 'find( sub {
      -e 'local $$_= $File::Find::name;''
      -e 'return unless /ims\d-\d+\.jpg$$/;''
      -e 'push @files, $$_;'"
      -e '}, $$dir );''
    -e 'print qq{<H2>FR Engine: $dir</H2>};'"
```
Usage: make

```
$ make
  ( echo "dd=[ ...";\n  bzip2  -dc cognitek-sdk199/test0/ims0-errcurve.txt.bz2 ;\n  echo " ];"; \n  echo " "
  dd( 1:420:length(dd),:)=[]; \n  eopen('``pwd`/cognitek-sdk199/test0/ims0-errcurve.doc'); \n  eglobpar; 
  eAxesLabelFontSize= 5;\n  ePlotAreaWidth= 50; \n  ePlotAreaHeight=60; \n  eAxesTicShortLength=0; \n  "AxesLabelWidth": 2, \n```
Results

web page created which summarizes generated images and graphs
Advantages

- Keeps short code in one place
- Automatic dependencies
  - Large jobs logically break up into smaller ones. Debugging can focus on pieces of job
  - Only works if dependencies are files. No Databases, for example
- Mix languages
- File format, easy to version control