scmdgen

Small Command Generator

`scmdgen` (pronounced 'smidgen') takes a base set of commands and expands it as specified by the user. It is designed to work with Salford Predictive Modeler (SPM), but in theory, it should work with any programming language, or even with generic text files that have nothing to do with computer programming.

Prerequisites

`scmdgen` is a Perl script. The following modules are required:

- boolean
- Getopt::Long
- Scalar::Util
- Set::Tiny

What it does

Consider the following command sequence:

```
HEAD output scmdgen_demo
HEAD note "Count from one to ten"
note "N"
TAIL quit
```

`scmdgen` can transform this into an SPM program that counts from 1 to 10, like so:

```
output scmdgen_demo
note "Count from one to ten"
note "1"
note "2"
note "3"
note "4"
note "5"
note "6"
note "7"
note "8"
note "9"
note "10"
quit
```

In the above example, any lines starting with `HEAD` are output first, those starting with `TAIL` are output last, and the remaining lines are repeated with `N` being replaced by integers from 1 to 10.

Assuming the original set of commands reside in `scmdgen_demo.txt`, we would invoke `scmdgen` as follows:

```
scmdgen --input=scmdgen_demo.txt N=1:10
```

Instead of specifying a sequence, one could specify a set of arbitrary strings with which to replace the token. For example,
consider the following sequence (`bostn2a.txt`):

```
HEAD submit fpath
HEAD use boston
HEAD submit labels
HEAD category chas
HEAD model mv
output bostn2a_LOSSFUNC
grove bostn2a_LOSSFUNC
memo "Basic TN model on the Boston housing data"
memo "LOSS=LOSSFUNC"
memo echo
treenet loss=LOSSFUNC go
```

To build models varying the TreeNet loss function, one could invoke `scmdgen` as follows:

```
scmdgen --input=bostn2a.txt --output=bostn2a.cmd LOSSFUNC=LAD,LS,HUBER,RF
```

...after which, the contents of `bostn2a.cmd` would read:

```
submit fpath
use boston
submit labels
category chas
model mv
output bostn2_LAD
grove bostn2_LAD
memo "Basic TN model on the Boston housing data"
memo "LOSS=LAD"
memo echo
treenet loss=LAD go
output bostn2_LS
grove bostn2_LS
memo "Basic TN model on the Boston housing data"
memo "LOSS=LS"
memo echo
treenet loss=LS go
output bostn2_HUBER
grove bostn2_HUBER
memo "Basic TN model on the Boston housing data"
memo "LOSS=HUBER"
memo echo
treenet loss=HUBER go
output bostn2_RF
grove bostn2_RF
memo "Basic TN model on the Boston housing data"
memo "LOSS=RF"
memo echo
treenet loss=RF go
```

The resulting command file `bostn2a.cmd` could then be submitted to SPM and the four requested TreeNet models would be built.

If one wanted to write the code to build each model to a separate command file, one could use the `--baseout` flag, like so:

```
scmdgen --input=bostn2a.txt --baseout=bostn2a LOSSFUNC=LAD,LS,HUBER,RF
```

This would create output files `bostn2a0.cmd`, `bostn2a1.cmd`, `bostn2a2.cmd`, and `bostn2a3.cmd`. To use the actual
values in the names, one could specify the `--use-values` flag, like so:

```
scmdgen --input=bostn2a.txt --baseout=bostn2a --use_values
LOSSFUNC=LAD,LS,HUBER,RF
```

This would instead create output files `bostn2a_LAD.cmd`, `bostn2a_LS.cmd`, `bostn2a_HUBER.cmd`, and `bostn2a_RF.cmd`.

The command stream could be submitted to SPM by `scmdgen` itself as follows:

```
scmdgen --input=bostn2a.txt --exec=spmu LOSSFUNC=LAD,LS,HUBER,RF
```

One can also specify multiple `scmdgen` variables (tokens), like so:

```
scmdgen --input=2digit.txt --output=2digit.cmd N=0:9 M=0:9
```

In this case, all possible combinations of `N` and `M` would be used.

One could also limit the number of combinations generated, like so:

```
scmdgen --input=2digit.txt --output=2digit25.cmd --ncombo=25 N=0:9 M=0:9
```

In this case, 25 combinations, selected at random, would be used.

To use all odd values of `N` and all even values of `M`:

```
scmdgen --input=2digit.txt --output=2digitoe.cmd N=0:8:2 M=1:9:2
```

In this case, 2 is the amount by which to increment the two variables.

One can pipe the output to a SPM like so:

```
scmdgen --input=bostn2a.txt LOSSFUNC=LAD,LS,HUBER,RF | spmu
```

But one can also use the `--exec` flag to do it directly like this:

```
scmdgen --input=bostn2a.txt --exec=spmu LOSSFUNC=LAD,LS,HUBER,RF
```

And because scmdgen will work with any program that accepts data from standard input, one can even send the output to a pager,
like so:

```
scmdgen --input=bostn2a.txt --exec=most LOSSFUNC=LAD,LS,HUBER,RF
```

Multiple Combination Sets

While it is often convenient to be able to request all combinations of a particular set of parameters, or a randomly drawn subset thereof, it is often desirable to vary one parameter while keeping the others constant, or to vary two or more parameters in concert with each other. `scmdgen` supports such cases by allowing the user to specify multiple sets of combinations, separated by "/". The slashes are separate arguments and muse be separated from the neighboring arguments by spaces. For example, consider the following base command file (`bostn2b.txt`).

```
HEAD submit fpath
HEAD use boston
HEAD submit labels
HEAD category chas
HEAD model mv
output bostn2bnNNODES
grove bostn2bnNNODES
memo "Basic TN model on the Boston housing data"
memo "NTREES NNODES node trees"
memo echo
treenet learnrate=.01 trees=NTREES nodes=NNODES go
```

To vary `NNNODES` and `NTREES`, keeping the product constant, we can invoke `scmdgen` as follows:

```
scmdgen --input=bostn2b.txt --output=bostn2b.cmd \
         NNODES=2 NTREES=6000 / NNODES=4 NTREES=3000 / NNODES=6 NTREES=2000 / \
         NNODES=8 NTREES=1500```
```

To vary `TREENET SUBSAMPLE` and `TREENET INFLUENCE` separately, holding the other constant, and piping the output to SPM for execution, one could invoke `scmdgen` like so:

```
scmdgen --input=bostn2c.txt --exec=spmu \
        SUBSAMPLE=.2:1:.2 INFLUENCE=.1 / SUBSAMPLE=.5 INFLUENCE=0:.6:.1
```

Delimited Variable Names

To avoid confusion with literal text, it is sometimes desirable to surround variable names with a delimiter string. For example, if the delimiter string is '%', the variable 'I' would be represented as '%I%'; conversely, with the delimiter removed, 'I' would be taken as literal text and thus would be unchanged. To specify such a delimiter (which could be, but probably shouldn't be, more than one character), use the `--dlm` flag.