MEMO: Using UNIX shell commands to recode long identifiers, with application to STATA

Andrew Noymer*

7 September 2004

1 The Problem

Many datasets contain unique identifiers (UIDs) that are very large integers (e.g. 10 digits or more). When read into STATA, these are stored as `double`, a floating-point type. Floating-point is not suitable as a UID because there is no exact floating-point comparison.

These long UIDs are almost always the result of administrative record-keeping effects, not of having billions of unique records. Thus, it is possible to create a new UID that can be stored as an integer type. With the Union Army data set (`www.cpe.uchicago.edu/unionarmy/unionarmy.html`), I am obtaining different sub-sets of the data on different occasions, but UIDs are comparable across all the various data extractions, so the approach needs to be consistent and reproducible. However, the problem of long UIDs is not unique to the Union Army data set; I have encountered it before in other settings.

❧

The purpose of this memo is three-fold. First, to document (for myself) the approach I have taken so that my results will be reproducible and for my own records generally. Second,

*andrew@demog.berkeley.edu

to share with others who may benefit, as a sort of How-To. Third, to solicit feedback on how this may be improved/streamlined — I feel like I am re-inventing the wheel, yet I have not found anything written on this.

2 Possible solutions to the problem

1. Use a bigger storage type.

   STATA's largest integer storage type at this time (version 8.2) is a signed 32-bit integer (maximum: 2,147,483,620). So this is not an option. Other programs I use, for example Kylix (Borland's Pascal dialect), support 64-bit integers (`Int64`), and it is hoped that STATA will incorporate this in the future.

2. Decipher the "administrative logic" behind the large UIDs.

   There may be redundancy in the UID such that (e.g.) the first four digits may be stripped-off without ruining the uniqueness of the UID. This is not usually intellectually very difficult — but is prone to careless errors that can be difficult to detect ex-post. The problem is that the data to be added later are not seen yet, so the "administrative logic" cannot be tested fully.[1]

3. Use strings.

   Convert the large integers to strings. This may be done within STATA (using `tostring`). Strings, like integer types but unlike floating-point, may be compared to each other exactly. This is an inelegant solution in my opinion because it is inefficient storage-wise and it seems unstylish to use

[1] To their great credit, the keepers of the Union Army data provide a breakdown on how their long UIDs are constructed, so "decipher" may not be the right word in all cases, but the intent of my current approach is to avoid careless errors that may arise even from misinterpreting a provided UID scheme.

strings as UIDs. Also, it may be impossible to loop over strings in STATA (?), and UIDs are sometimes looped.

4. Create a new UID.

   This is the path I have elected and the remainder of this memo shows how to do it.

3 Conceptual approach

Throughout this document, `origid` refers to the original (long) UID and `newid` refers to the new (short) UID.

This is a two-stage approach because my tasks with the Union Army data set involve getting data to begin with and then adding more data as time goes by. When dealing with a stand-alone data set, the first stage alone is sufficient.

Stage One (original data set):

1. Make a master list of unique `origid` values.
2. Assign a `newid` to each `origid` to create a match list.
3. Use the match list to insert `newid` into the raw data set.

Notes: (i) Do *not* delete the match list (a file) at the end of Stage One — it will be needed later. (ii) It will usually be desirable to hold on to `origid`, just for good measure.

Stage Two (adding more data later):

1. Cross-check the `origid` of the new records against the existing match list, and generate a temporary list of new `origid`.
2. Generate a temporary new match list from the outcome of the previous step (NB: the numbering must continue from Stage One).

3. Append the temporary new match list from the previous step to the master match list (from Stage One). This is the new master match list.

4. Repeat Stage One, step 3, using the new master match list and the new raw data.

5. In STATA, read in the new raw data, then append or merge with the old data.

This seems like a lot of fuss, but each step reasonably simple and straightforward (discussed in §4) and combining the steps is easy (discussed in §5).

4 Practical approach I: doing each step

My philosophy is to keep things simple. All this can probably be done within STATA. However, the Union Army Data set is not supplied to me in STATA format; it's supplied as tab-delimited text. So I find it easier to deal with the UID problem in the raw format, before reading the data into STATA.

This is run only once, then once again every time data are added. Therefore I do not see the need to spend a lot of human time to super-optimize the run time. These UNIX sort (etc.) commands are generally not slow. However, I would be grateful if readers of this document would point out blatant inefficiencies or complicated things that could be replaced by one-liners (other than a string of pipes a mile long).

The original data file is called `data.txt`, and `origid` is in the 22nd column (tab-delimited).

[*The final code has been streamlined a little — see § 5 for details. This section will be modified to reflect the changes as time permits.*]

Stage One (original data set):

1. Make a master list of unique `origid` values.

- first copy the UID column out of the data. awk is a natural choice but this works also:

  ```
  cut -f22 data.txt > uid.temp
  ```

- then remove duplicates:

  ```
  sort -n uid.temp | uniq > uid.temp2
  ```

2. Assign a `newid` to each `origid` to create a match list.

   - first number the lines of the previous output (if one wishes to avoid single-digit UIDs, start at 1000 or so; it doesn't matter):

     ```
     nl -nrn -p -v1000 uid.temp2 > master.list
     ```

3. Use the match list to insert `newid` into the raw data set. This is the trickiest part of Stage One. The master list, just generated, is a Rosetta stone between the `origid` and `newid`. This is converted into a sed script, the obvious tool for text substitution. But we cannot just go blasting through *all* of the original file because only the UID values are to be substituted — the chance of a false positive is high, especially in cases where the original data are fixed-format not tab-delimited. So we work with a copy of `origid`, convert it to `newid`, then paste it back in place. (It is imperative that nothing be done to change the sort-order or else we're in for big trouble.) As an alternative, awk could be used here, eschewing sed. I think of awk whenever I think of columnar data, but my knowledge of awk is mostly dealing with one open file, and here we are working with two files: the data, and the master list. I think what follows works well enough, and I don't know Perl or ksh, so this is what I could come up with.

   - first translate the master list into a sed script:

     ```
     awk 'print "s/" $2 "/" $1 "/"' master.list > temp.sed
     ```

- then copy the `origid` (as before):

  ```
  cut -f22 data.txt >| uid.temp
  ```

- then change the `origid` to `newid`:

  ```
  sed -f temp.sed uid.temp >| uid.temp.2
  ```

- then paste things back into place:

  ```
  paste data.txt uid.temp.2 > data.raw
  ```

Stage Two (adding more data later):

The data file to be added is called `data2.txt`, and `origid` is still in the 22nd column (tab-delimited).

1. Cross-check the `origid` of the new records against the existing match list, and generate a temporary list of new `origid`.

   - generate a list of the `origid` like in Stage One:

     ```
     cut -f22 data2.txt > uid.temp
     ```

   - generate a list of `origid` that we have already "cataloged":

     ```
     cut -f2 master.list > uid.ml.temp
     ```

   - generate a list of `origid` in the new file not in the old file:

     – things must be sorted (for `comm`):

       ```
       sort -n uid.temp | uniq > uid.temp2
       ```

       this should already be sorted, but for good measure:

       ```
       sort -n uid.ml.temp > uid.ml.temp2
       ```

     – the headers are a nuisance at this point:

       ```
       sed '1d' uid.ml.temp2 >| uid.ml.temp
       sed '1d' uid.temp2 >| uid.temp
       ```

     – then we get the list of things in the new file not in the old:

       ```
       comm -13 uid.ml.temp uid.temp > uid.new
       ```

2. Generate a temporary new match list from the outcome of the previous step (NB: the numbering must continue from Stage One).

   - first figure out where the numbering should begin:

     ```
     count=`tail -1 master.list | cut -f1`
     count=`expr $count + 1`
     ```

   - number the new list of `origid`:

     ```
     nl -nrn -p -v$count uid.new > new.master.list
     ```

3. Append the temporary new match list. . . :

   - ```
     cat master.list new.master.list > master.list.2
     /bin/mv master.list.2 master.list
     ```

4. Repeat Stage One, step 3, using the new master match list and the new raw data.

   - as before.

5. In STATA, read in the new raw data, then append or merge with the old data.

   - in the usual way. . .

5 Practical approach II: putting it all together

The following shell scripts provide "one-stop shopping" for everything above, up to but not including reading the data into STATA. The scripts include some details, such as removing un-needed temporary files, that were omitted above.

Stage One (original data set):

```
#!/bin/bash
#
```
```

```
# shell script to generate smaller but still-unique ID numbers
#
# version 0.1, August 2004
# Andrew Noymer
#
# detailed explanation in the accompanying memo (newid-memo.pdf)
#
# (could get fancy and read these from the command line etc...)
rawdata=data.txt
output=`basename $rawdata .txt`
# should read this by parsing the header:
UIDcolno=22
# The first newUID will be this plus 1:
startUID=999

# beware using "sort -nu" on multicolumnar data -- it doesn't come up here, but there appears
# to be a bug in this; use "sort -n | uniq".  for single-column data "sort -nu" is OK

cut -f$UIDcolno $rawdata > uid.temp
header=`head -1 uid.temp`
newheader=new$header
sort -nu -ouid.temp uid.temp
nl -nrn -p -v$startUID uid.temp > master.list.temp
sed '1 s/'$startUID'/'$newheader'/' master.list.temp > master.list
awk '{print "s/" $2 "/" $1 "/"}' master.list > temp.sed
cut -f$UIDcolno $rawdata | sed -f temp.sed >| uid.temp2
paste data.txt uid.temp2 > $output.raw
/bin/rm temp.sed uid.temp uid.temp2 master.list.temp
chmod 444 master.list
```

Stage Two (adding more data later):

```
#!/bin/bash
#
# shell script to generate smaller but still-unique ID numbers
# when adding data to an already-existing dataset
#
# version 0.1, August 2004
# Andrew Noymer
#
# detailed explanation in the accompanying memo (newid-memo.pdf)
#
# (could get fancy and read these from the command line etc...)
rawdata=data2.txt
output=`basename $rawdata .txt`
# should parse the header to determine this
UIDcolno=22
```

```
priormasterlist=master.list

# beware using "sort -nu" on multicolumnar data -- it doesn't come up here, but there appears
# to be a bug in this; use "sort -n | uniq".  for single-column data "sort -nu" is OK

chmod 644 $priormasterlist
cut -f$UIDcolno $rawdata > uid.temp
cut -f2 $priormasterlist > uid.ml.temp
sort -nu uid.temp > uid.temp2
# the following should already be sorted... prob. should remove this line
sort -n uid.ml.temp > uid.ml.temp2
sed '1d' uid.ml.temp2 >| uid.ml.temp
sed '1d' uid.temp2 >| uid.temp
comm -13 uid.ml.temp uid.temp > uid.new
# perform some of the following only if uid.new is not empty
if [ -s uid.new ]
 then
{
  count=`tail -1 $priormasterlist | cut -f1`
  count=`expr $count + 1`
  nl -nrn -p -v$count uid.new > new.master.list
  cat $priormasterlist new.master.list > master.list.2
  /bin/mv master.list.2 $priormasterlist
  /bin/rm new.master.list
}
fi
awk '{print "s/" $2 "/" $1 "/"}' $priormasterlist > temp.sed
cut -f$UIDcolno $rawdata >| uid.temp
sed -f temp.sed uid.temp >| uid.temp2
paste $rawdata uid.temp2 > $output.raw
/bin/rm uid.new uid.temp uid.ml.temp2 uid.temp2 temp.sed uid.ml.temp
chmod 444 $priormasterlist
```

6 Known bugs

If the data files are padded with blank lines before the `EOF`, I think it messes everything up.

7 Keeping the master list file

The match list file should be retained as long as more data will be added. But as long as `origid` has not been discarded in the data, it should be possible to reverse-engineer everything in the event that the match list file is deleted.

8 Postscript: further possibilities

I have encountered alphanumeric UIDs in the past and dealt with these using sed to convert letters to numbers, and then a variation of the above. Care must be taken if the UIDs are a mix of pure numeric and alphanumeric, so that, e.g., 1Z11 does not get converted to 12611, because UID 12611 may already exist.