Notes on the History and Design of UNIX|STAT

Gary Perlman

1. INTRODUCTION

In this memo, I discuss the issues in the design of the UNIX|STAT programs. UNIX|STAT is a collection of programs running on the UNIX* operating system. The programs are designed to be easy to use in the sorts of data analysis problems common in analyzing experimental psychological data. Because data analysis is a common activity in many fields, the programs have found use in a variety of contexts. To date, the programs are being distributed to over 100 UNIX sites in 11 countries on 4 continents, and with their recent release on the USENIX distribution tape, their use will probably increase. Most UNIX|STAT users have access to other more powerful statistical packages, but for the cases where UNIX|STAT programs are applicable, the programs seem to be preferred because they are so easy to use. In later paragraphs, the reasons for their ease of use will be discussed.

2. HISTORY

UNIX|STAT began when I was a psychology graduate student at the University of California, San Diego. We had Version 6 UNIX running on a DEC PDP 11/45 which had limited memory. As a psychological laboratory, we had data to analyze, but we had no analysis programs on UNIX. Occasionally, we got a tape from a site with their latest programs, but they tended to be too big, too slow, too clumsy, or not do what we wanted them to do. I tried to find statistical packages with no success. We even got a standard large statistical package, but its programs were not designed for our little PDP 11/45. So huge were they, with all their immense power, that to load a program and do all the necessary overlaying of subroutines required two minutes to get the mean of one number. We were not very happy.

I wrote the first version of DESC about that time; it was a program to describe a single distribution of data with summary statistics, frequency tables, and histograms. I was tired of writing a *mean* program every time I wanted to do the simplest data analysis, so I tried to write a general descriptive once and for all. DESC was not welcomed with much fan fare; I had to go door to door to get people to use it. Evidently, people thought it would be easier to write their own programs, or do the analyses with a calculator, then learn ot use my program. I searched personal binary directories to see if people were making their own programs to do sub-parts of DESC. Sure enough, people had programs, even shell programs, to compute means and the occasional standard deviation. Their programs were difficult to use, with no documentation and silly conventions like having to end the input with a minus one. DESC allowed free format input and read until the end of a file, provided many options, and its output was easy to read. After about six months, DESC was in moderate use by part of our lab. I was encouraged to see people other than myself berating a third party for writing their own programs to do what DESC did.

After DESC, I wrote PAIR, a program for paired data analysis (a common design in psychology). PAIR was based on a program written by Don Gentner of our lab. I liked his program, but not its output format, which I modified. PAIR followed similar format conventions as DESC and people familiar with DESC found PAIR that much easier to use. Over the years since then, additional statistics and even a simple scattergram routine have been added.

Both DESC and PAIR read from the standard input and write to the standard output. The main reason for this was that it is easier than opening files. Looking back on this convenient decision, I see it as critical to the design of the rest of the package. The standard UNIX philosophy is to to create modular

---

* UNIX is a trademark of Bell Laboratories.

programs that can be connected via pipelines with existing tools in novel ways. This is worth repeating. **The standard UNIX philosophy is to to create modular programs that can be connected via pipelines with existing tools in novel ways.** I knew of this philosophy, but did not follow it for any particular reason. As other UNIX|STAT programs developed, particularly ones for transforming data, the philosophy dominated the design.

We tended to keep our data in matrix type files with some uniform number of blank separated fields per line. We used these as master data files from which we extracted the columns we were interested in. Jay McClelland wrote a program called *colex* which extracted columns from such files. *Colex* was indispensable, and Jay added features to extract columns based on conditions by allowing analysts to set minimum and maximum allowable values for each column. We wanted to be able to add, subtract, and do other transformations on the values of the columns to create new files of data so I wrote a program called *trans* which did an inadequate job using a simple syntax. We needed some data massager that could be handed complex expressions.

In the summer of 1980, I was teaching myself about compilers and at the same time learned of a "compiler-compiler" called *yacc*. Based on our need for a data massager and my interest in *yacc*, I wrote DM, now called the data manipulator for public consumption. DM allows an analyst to write a series of C-like expressions for conditional extraction of algebraic combinations of columns in a master data file. With DM, *colex* and *trans* became immediately obsolete, especially after Jay helped add DM's string handling capabilities. It was not until a year later that we found out that there was a program called *awk* that could do all of what DM did, and more. Even so, DM held its own because it was easier to use, and for common tasks like column extraction, faster.

After DM was around, our view of data analysis changed to that of a series of transformations followed by a specific analysis.

cat data | transformations | analysis

Data begins in a master data file, is sent via a pipe to a series of transformations (e.g., by DM) followed by a pipe into an analysis program (e.g., DESC). Jay McClelland's concept of having all the data in a master data file, and a set of transformation routines prompted him to write the original version of ABUT on which mine is based. ABUT helps construct master data files while DM takes them apart. Dave Rumelhart wrote a program called *mc* (for Make Column) to re-shape data files; MAKETRIX is based on this. And finally, the TRANSPOSE program was added to the list of transformation tools.

At about the same time, Craig Will was working on a front end to some standard statistical package's anova program, the kind with unforgiving fixed column input formats (all UNIX|STAT programs use white-space separated fields). I thought this was noble, but that the design of such programs was not that easy to fix. Besides the input formats, the control languages were abominable, especially those parts to specify the experimental design. I decided that there must be a natural way to enter that data from a design in a way that made it impossible to make an error and while making the data easier to describe.

At about this time, Jay McClelland wrote his *dt* (for Data Tabulation) program to get cell means using an innovative input format. Each datum is accompanied by a set of identifying strings which are used by *dt* to decide what cell a datum belongs to. For example, if you found that a department store named ABC sold large shoes for $10.95, you would enter a line like:

ABC large shoe 10.95

This is the essence of the relational database model.

This concept of self identification was followed in ANOVA in which each datum is preceded by a series of labels, one for each factor, identifying the conditions of the factors under which the datum was collected. A file of data like this is self documenting, and ANOVA is able to infer the design from its format without any interactive specification by the user. The REGRESS program was designed to accept its natural input format too: a file of N columns implies an N variable regression. This is easy to explain to even the most computer naive analysts.

By this time, the package had reached a critical mass, at least for our laboratory's needs. Someone suggested I share these programs with other psychologists and sent a general description of them to a journal in which psychologists share technical information about *doing* research as opposed to the *results* of research. On Greg Davidson's suggestion, I added more extensive error checking to the programs before releasing them to the public.

3. INTERACTIVE USE

The UNIX shell is the command language for UNIX|STAT. Using the standard input and output in the programs allows joining them together with pipelines, and the shell provides the command language to do it. Using the shell is advantageous because it saves analysts learning time. If analysts are familiar with the shell and some UNIX programs like *ed*, *sort*, *mv*, *rm*, and others, then they will be on familiar ground. If not, they can be introduced to these utilities in a familiar context, and they will be able to use this knowledge in other contexts. The main point of having all transformation and analysis routines as stand alone programs and the shell as the command language is that they can be used with UNIX utilities by anyone familiar with UNIX. The opposite is also true: some of the transformation routines are used in contexts other than data analysis.

The IO program is an intelligent controller and monitor of input and output in the shell and is based on Don Norman's notions about flow of data through pipelines and the need to monitor it. Monitoring the progress of programs is important if complex calculations are being performed because they can take a long time, sometimes several minutes. IO allows an analyst to get feedback about how much of a source file has been exhausted, or how many blocks of data have passed through a pipe. With a complex pipeline of transformations, an irritating error is to forget to specify an input file. Besides IO *not* giving any feedback of progress, most of the UNIX|STAT programs inform the analyst when no input redirection is specified and data are to be read from the terminal keyboard. The programs assume that data entry from the keyboard is the exception rather than the rule. IO is also a safety minded replacement for input and output redirection because it does not allow analysts to accidently overwrite the contents of files.

UNIX|STAT assumes that user efficiency is more valuable than computer efficiency. The cost paid by having a slow algorithm (perhaps because a free-format Ascii data file is being processed) or larger than necessary data files (because they are self documenting) is easy to compute. The cost in wasted human resources because of increased command specification time, increased probability of error and increased correction time, and the high cost of undetected errors is more difficult. Rather than count time for a program to run, it is more reasonable to count the time an analyst spends constructing the commands to run the program and interpret the results. These ideas are most evident in the design of the ANOVA program, which removes as many time consuming and error prone tasks as possible from the analyst.

4. COMPARISON WITH S

I am often asked to compare UNIX|STAT to the S data analysis system, developed at Bell Labs by Rick Becker and John Chambers. UNIX|STAT is entirely independent of S and was developed without its knowledge. The major difference is that UNIX|STAT was developed for working with the sorts of experimental psychological data we in our lab at UCSD were accustomed to using, while S was developed as a general data analysis system, and so offers a more comprehensive set of functions. The philosophies of design of the two systems are similar in some ways, and very different in others.

UNIX|STAT is a set of programs that are *added* to the existing set of UNIX programs. They all can read and write using pipes, and so can be used in conjunction with most other UNIX utilities, using the facilities provided by the UNIX shell. For example, if you wanted to sort the data in a data file, you would use the UNIX *sort* program, not one provided by UNIX|STAT. Or if you wanted graphical output, you would use the standard UNIX *plot* and *graph* programs (though these could benefit by having a high

level front end). One reason why it is possible to use UNIX utilities is that UNIX|STAT programs assume a simple Ascii format file with data separated by white space (spaces, tabs, and sometimes newlines).

S is a special environment for doing data analysis separated from the UNIX environment. The programs use a special data format so, in general, they can not be used in conjunction with UNIX utilities. In general, it is not possible to use some standard UNIX utility with S data sets because the data have such a special format. One result of this is that S needs to have its own versions of many utilities already existing in UNIX. Another is that is is, in general, difficult at best to use S functions outside of the S environment.

Both systems are extensible, though I suspect adding to a set of UNIX programs is easier. Both systems offer programming capabilities: S has its own macro definition language while UNIX|STAT depends on the shell's programming language. Using one over the other depends on the needs of the analyst; certainly, S's set of statistical functions is much larger. Concerning efficiency, my impression, without benchmarks or controlled data, is that the UNIX|STAT programs are faster and easier to use for small to medium data sets, especially for one shot analyses. For larger data sets, S's matrix manipulation primitives seem to give it an advantage that makes up for its slow startup time. Concerning portability and related areas, the UNIX|STAT programs use 300 blocks for the C source files, and 500 blocks for the executables, each program averaging 14000 bytes. I am unsure about the requirements for S, but I know them to be much larger. The UNIX|STAT programs are running on UNIX systems dating back to Version 6 UNIX (of course, all subsequent systems can run them), and run on computers as small as PDP 11/23's without problem.

5. FURTHER INFORMATION

5.1 Availability

The UNIX|STAT programs have been released in the public domain. In general, the source files, documentation, and examples, are distributed at cost to any interested parties with the understanding that:

1. The programs are not to be sold for material gain.
2. The software, though tested over several years, is not guaranteed in any way, and certainly not supported.
3. UNIX|STAT is not a product, nor supported by Bell Laboratories.

5.2 Other UNIX|STAT Papers

1. Data analysis programs for the UNIX operating system. *Behavior Research Methods and Instrumentation*, 1980, **12**, 554-558.
2. An example of cooperating compact data analysis programs. *Behavior Research Methods and Instrumentation*, 1981, **13**, 290-293. Presented at the National Conference on the Use of On-Line Computers in Psychology, November 12, 1980, St. Louis, Missouri.
3. Data analysis in the UNIX environment: Techniques for automated experimental design specification. Presented at the XIV Annual Conference on the Interface, July 7, 1982, Rensselaer Polytechnic Institute, Troy, New York. Proceedings to be published by Springer-Verlag.
4. UNIX|STAT: Compact data analysis programs for UNIX. Presented at the 1982 Summer USENIX Conference, July 9, 1982, Boston, Massachusetts. Proceedings to be published by USENIX.

CONTENTS

1. INTRODUCTION 1
2. HISTORY 1
3. INTERACTIVE USE 3
4. COMPARISON WITH S 3
5. FURTHER INFORMATION 4
5.1 Availability 4
5.2 Other UNIX|STAT Papers 4

Notes on the History and Design of UNIX|STAT

Gary Perlman