© 1986 Gary Perlman

Section 3.1 Command Line Interpreters

|STAT analyses consist of a series of commands, each on a single line, hence the name *command line*. Commands are typed by users into a command line interpreter, itself a program that runs the commands typed in. On MSDOS, there is no special name given to the command line interpreter. On UNIX, the command line interpreters are called *shells*, and there are several of them. Users are expected to know the conventions of their command line interpreters. Some of the examples in this handbook and in the manual entries will not work because of differences in how command lines are formatted. Minor modifications to the examples are sometimes needed.

Some command line interpreters support in-line editing, which is useful when running |STAT analyses because data analysis is an iterative process in which minor changes in analyses, and hence commands, are common.

Special Characters

Command line interpreters have special characters to perform special tasks. On both MSDOS and UNIX, there are special characters for file input, output, and pipe redirection:

<	redirect standard input from the following file
>	redirect standard output to the following file
|	redirect standard output to the following command

UNIX and MSDOS both have patterns (sometimes called ‘‘wildcards’’) to match file names. For example, ***.c** matches all files that end with a **c** suffix. Also, the **?** can be used in patterns to match any one character. An important difference between UNIX and MSDOS command line interpreters is that on UNIX, the pattern matching is part of the shell, and so is available to every program, while on MSDOS, it is part of only some programs.

It is sometimes necessary to quote the special meaning of special characters so that they are not seen by the command line interpreter. For example, an expression for **dm** might contain the symbols ***** for multiplication or **<** for comparison. Both these characters are special to UNIX shells, while only **<** is special to MSDOS. The blank space and tab characters are special on both UNIX and MSDOS, and are used to separate command line arguments. Special characters can be quoted by enclosing command line arguments in double quotes. For example, **dm** expressions may contain special characters, and strings may contain spaces.

```
dm  "if x1 > 10 then 'Large number on line:' else SKIP"  INLINE
```

© 1986 Gary Perlman

Section 3.2 Command Formats

|STAT programs are run on UNIX and MSDOS by typing the name of the program, program options, and program operands (e.g., expressions or file names). Program names, options, and operands, are separated or *delimited* by blank space. On UNIX, program names are lower case, while on the case-insensitive MSDOS, they are always upper case, although users can type the names in lower case. Program options and operands can be complex, so it is sometimes useful to insert spaces into an option value or an operand, either to modify the output or to make the command line more readable. This is done by quoting (with double quotes) the parts that should be kept together.

Simple Commands

A simple command consists of a program name, program options delimited with minus signs, and program operands, such as file or variable names. Here are some examples:

```
dm  x1+x2  x3/x4
calc  model
regress  -p  age  height  weight
desc  -h  -i 1  -m 0  -cfp
series  1  100  .5
probdist  random  normal  100
```

Pipelines of Commands

A pipeline of commands is a series of simple commands joined by the pipe symbol, **|**. In a pipeline, the output from one simple command is the input to the next command in the pipeline. The following pipeline creates a series of numbers from 1 to 100, transforms it by using the **dm** logarithm function, and then makes a histogram of the result.

```
series 1 100  |  dm logx1  |  desc -h
```

The following pipeline abuts three files beside one another, and passes the result to the **regress** program, which prints their correlation matrix.

```
abut age height weight  |  regress -r age height weight
```

Note that the operands to **abut** are file names, while those for **regress** are variable names, which could be different if desired. If they were always supposed to be the same, then this constraint could be encoded in a shell script or batch file.

Batch Files and Shell Scripts

Because the |STAT programs work well together, and because most data analysis is routine, it is often advantageous to save a series of commands in a file for later analyses. Both UNIX and MSDOS support this, MSDOS with *batch* files and UNIX with *shell* scripts. Batch files and shell scripts also support variables, some set by command line calls and some set inside the command file. They provide |STAT with a simple but effective programming facility.

© 1986 Gary Perlman

Section 3.3 Program Options

Program options allow the user to control how a program works by requesting custom or extra analysis. Without options, |STAT programs provide the simplest or most common behavior by default. Program options conform to the standard UNIX option parsing convention (Hemenway & Armitage, 1984) by using the **getopt** option parser. In this standard, all program options are single characters preceded by a minus sign. For example, **-a** and **-X** are both options. All program options must precede operands (such as file names, variable names, or expressions). Some options require values, and these should follow the option. For example, the **pair** plotting function allows setting the height of the plot with the **-h** option: **-h 30** would set the plot height to 30 lines. There should be a space between an option and its value. Options that do not take values (logical options) can be grouped or ''bundled'' to save typing. For example, the descriptive statistics program, **desc**, has options for requesting a histogram, a table of frequencies, and a table of proportions. These can be requested with the bundle of options: **-hfp** instead of the longer: **-h -f -p**.

There are some special conventions used with the **getopt** option parser. A double dash, **--**, by itself signals the end of the options, which can be useful when the first operand begins with **-** and it would be misinterpreted as an option. For programs that take files as operands (e.g., **abut**, **calc**), a solitary **-** means to read from the standard input, which can be useful to insert the output of a pipeline in a set of files. For example, the **abut** program can read several files with the standard input inserted with the following command line.

```
series 1 20  |  abut file1 file2 - file3
```

The output would be four columns, the third of which would be the series 1 to 20.

The same options can usually be specified more than once on a command line. For logical options (those that turn on or off a feature), repetition usually has no effect. For options that take values, such as the width of a plot, respecifying an option resets it to a new value. Exceptions to these rules for specific options are mentioned in program manual entries.

Table of Option Rules

-x	options are single letters preceded by minus
-h 30	option values must follow the option after a space
-nve	logical options can be bundled
--	signals the end of the options
-	insert standard input to operands of file-reading program

Standard Options

All |STAT programs using the standard option parser, **getopt**, have standard options to get information online. The information reported by the program is always accurate, while the printed documentation may not be up to date, or may not apply to the particular version (e.g., limits on MSDOS may be smaller than on UNIX).

-L	prints a list of program limits
-O	prints a summary of program options
-V	prints version information

© 1986 Gary Perlman

Section 3.4 File Inputs and Outputs

Most of the |STAT programs are filters. That means they read from the standard input and write to the standard output. By default, the standard input is the keyboard, and the standard output is the screen. The standard input and output can independently be ''redirected'' using the special characters: **<**, to redirect the standard input from an immediately following file name, **>**, to redirect the standard output to a file. Also, the pipe character **|**, can connect the output from one program to the input to another. (Some of these features are not available on early versions of MSDOS (before version 2.0).) The following command says for the **anova** program to read from the file **anova.in**.

```
anova  <  anova.in
```

The output would go to the screen, by default. The following command saves the above output to the file **anova.out**.

```
anova  <  anova.in  >  anova.out
```

Never do this:

```
anova  <  data  >  data          # Never Do This!
```

Never make the input file the same as the output file, or you will lose the file; the output file is created (and zeroed) by the command line interpreter before the input file is read. Temporary files should be used instead. Here is an example of output redirection to save 50 random normal numbers.

```
probdist random normal 50  >  numbers
```

In English, this is read: ''A random sample of 50 numbers is created and saved in the file **numbers**. This file of numbers could be used as input to the descriptive statistics program, **desc**. The intermediate file, **numbers**, could be avoided by using a pipeline.

```
probdist random normal 50  |  desc
```

To save the result of the above analysis in a file called **results**, output redirection would be used.

```
probdist random normal 50  |  desc  >  results
```

Although pipes are supported on MSDOS, they are not efficient and they require that there is enough space for temporary files to hold the contents of the pipes (temporary files with names like **PIPE%1.$$$**). This can make input and output redirection without pipes a better choice for speed, especially in command scripts, called ''batch files'' on MSDOS.

Keyboard Input

If a program is expecting input from the keyboard (ie. the standard input has not been redirected from a file or pipe), a prompt will be printed on the screen. Often, input from the keyboard is a mistake; most people do not type directly into an analysis program but prepare a file with their preferred editor and use that file as input.

```
prompt: desc
desc: reading input from terminal:
user types input, followed by end of file: ^D on UNIX, ^Z on MSDOS
```

In all examples of keyboard input, the sequence **^X** will be used for control characters like control-x (hold down the **CTRL** key and type the letter **X**). On UNIX, end of input from the keyboard is signaled by typing **^D**. MSDOS users type **^Z**.

© 1986 Gary Perlman

Section 3.5 Input Formats

|STAT programs have simple input formats. Program input is read until the end of file, *EOF*, is found. End of file in disk files is done by the system; no special marking characters are needed nor allowed.

Input **fields** (visibly distinguishable **words**) are separated by **whitespace** (blank spaces, tabs, newlines). For most programs, fields in lines with embedded spaces can be enclosed by single or double quotes. Most |STAT analysis programs ignore blank input lines used to improve the human-readability of the data. However, blank lines are meaningful to some data manipulation programs, so when there are unexpected results, it is often instructive to run a file through **`validata`**.

Suggestion: Staged Analysis

It is usually a good idea to build a complex command, such as a pipeline, in stages. At each stage, a quick visual inspection of the output catches most errors you might make.

Data Types

|STAT programs recognize several types of data: label and variable names, numbers (integers and real numbers), and some programs can deal with missing values, denoted by **`NA`**. Label and variable names begin with an alphabetic character (a-z or A-Z), and can be followed by any number of alphanumerics (a-z, A-Z, 0-9) and underscores. There are three types of numbers: integers, real numbers with a decimal point, and numbers in exponential scientific notation. Integers are positive or negative numbers with no decimal point, or if they have a decimal point, they have no non-zero digits after the decimal point. Exponential notation numbers are numbers of the form **`xxx.yyyEzz`**. They may have digits before an optional decimal point or after it, and the number after the **`E`** or **`e`** is a power of ten multiplier. For example, **`1.2e-6`** is 1.2 times the inverse of one million.

Caveat: Appearances Can Be Deceiving

Inputs that look like they line up might not appear so to |STAT programs. For example, the following data might appear to have four columns, but have a variable number. Also, the columns that look like they line up to a person, do not line up to |STAT programs.

```
a     b     c     d
e           f     g
h     i           j
```

Here is how |STAT programs see this input:

```
a     b     c     d
e     f     g
h     i     j
```

This difference could be found with the **`validata`** utility program, which would report for both formats above:

```
validata: Variable number of columns at line 2
Col   N  NA alnum alpha   int float other  type   min   max
  1   3   0     3     3     0     0     0 alnum     0     0
  2   3   0     3     3     0     0     0 alnum     0     0
  3   3   0     3     3     0     0     0 alnum     0     0
  4   1   0     1     1     0     0     0 alnum     0     0
```

© 1986 Gary Perlman

Section 3.6 Limits and Error Messages

There is a system-dependent limit on the count of characters in an input line: on small systems, 512 characters, and on large ones, 1024. Many programs use dynamic memory allocation so the memory available on a machine will determine the size of data sets that can be analyzed. Integer overflow is not checked, so numbers like data counts are limited on 16 bit machines to 32767; in practice, this has not presented problems. All calculations are done with double precision floating point numbers, but overflow (exceeding the maximum allowed double precision number, about 10 to the 38th power) and underflow (loss of precision of a tiny non-zero result being rounded to 0.0) are not checked. Program specific limits can be found in most programs with the **`-L`** option. The programs are not robust when used on highly variable data (differences of several orders of magnitude), very large numbers, or large datasets (more than 10,000 values).

All error and warning messages (1) identify the program detecting the problem (useful when pipelines or command scripts are used), (2) print diagnostic information, (3) sound a bell, and for errors, (4) cause an exit. All error and warning messages are printed on the diagnostic output (that is **`stderr`** for C lovers), so they will be seen even if the standard output is redirected to a file. All |STAT programs exit with a non-zero exit status on error and a zero exit status after a successful run.

Common Error Messages

Some errors and messages are common to several programs. They are explained below. Other messages should be self-explanatory.

`Not enough (or no) input data`
: There were no data points read, or not enough to make sense

`Too many xxxx's; at most N allowed`
: Too many of something were in the input (e.g., columns or variables)

`Cannot open 'file'`
: The named file could not be opened for reading

`No storage space left for xxxx`
: The program has run out of dynamic memory for internal storage

`´string' (description) is not a number`
: The described object whose input value was 'string' was non-numerical

`N operand(s) ignored on command line`
: Operands (e.g., files) on the command line are ignored by this program

`VALUE is an illegal value for the TYPE`
: The provided value was out of the legal range for the given type

`Ragged input file`
: The program expects a uniform number of input columns

© 1986 Gary Perlman

Section 3.7 Manual Entries

|STAT manual entries contain detailed information about each of the programs. They describe the effects of all the options.

On-Line Manuals

On UNIX systems, the manual entries for |STAT programs are available online with the **manstat** program. UNIX system administrators might prefer to install the |STAT manuals in a public place, so they might be available with the standard UNIX **man** program. On MSDOS systems, manual entries might be available online with a batch file that types pre-formatted manuals. The following will print the online manual for the **anova** program.

```
manstat anova
```

Most programs print a summary of their options with the **-O** option. The following will print a summary of the options available with the **desc** descriptive statistics program.

```
desc -O
```

UNIX Manual Conventions

UNIX manual entries are often considered cryptic, especially for new users. It helps to know the conventions used in writing manual entries. In the following table, the contents of the different manual entry sections are summarized.

ALGORITHMS
: sources or descriptions of algorithms

BUGS
: limitations or known deficiencies in the program

DESCRIPTION
: details about the workings of the program,
and information about operands

EXAMPLES
: examples of command lines showing expected use of the program

FILES
: files used by the program (e.g., temporary files)

LIMITS
: limits built into the program should be determined with the -L option

NAME
: the name and purpose of the program

OPTIONS
: detailed information about command line options (see the -O option)

SYNOPSIS
: a short summary of the option/operand syntax for the program
(items enclosed in square brackets are optional)

© 1986 Gary Perlman

CHAPTER 3

Conventions

3.1 **Command Line Interpreters** 3-1
Special Characters
3.2 **Command Formats** 3-2
Simple Commands
Pipelines of Commands
Batch Files and Shell Scripts
3.3 **Program Options** 3-3
Standard Options
3.4 **File Inputs and Outputs** 3-4
Keyboard Input
3.5 **Input Formats** 3-5
Suggestion: Staged Analysis
Data Types
Caveat: Appearances Can Be Deceiving
3.6 **Limits and Error Messages** 3-6
Common Error Messages
3.7 **Manual Entries** 3-7
On-Line Manuals
UNIX Manual Conventions

Features common to all the |STAT programs are covered. This information makes it easier to learn about new |STAT programs, and serves as a reference for experienced users.

© 1986 Gary Perlman