Bell Laboratories

subject: **UNIX Command Syntax**

Filing Case 40125-001

date: **February 16, 1979**

from: **A. S. Cohen**
MH 2524
2F-223 x6920

S. B. Olsson
MH 2524
2C-253 x3474

G. C. Vogel
MH 2524
2C-158 x6115

2524-79-0216-02MF

ABSTRACT

The current UNIX™ command-line syntax is riddled with inconsistencies. Command-syntax rules and a library routine for achieving consistent syntax are proposed.

MEMORANDUM FOR FILE

There are now thousands of Bell Labs employees using UNIX™ time sharing systems. Many of these people are occasional users who are unfamiliar with the internal workings of the system, the many features of the shell, the historical development of the system, or the quirks of the command syntax.

Consider the following command lines:

```
lpr −l −m wlist
lpr −cm zlist
tp tm1 gold
pr −h "Grocery List" −5 −w100 glist
```

A user might reason from the first two lines that options may be grouped for convenience. The third line suggests a command syntax in which the command modifiers (options or keys) need not be prefixed by a '−'. Finally, the last line displays a mixture of options taking arguments. The 'h' option allows white space before its argument, the 'w' option allows its argument to be adjacent, and the '−' takes its argument without white space. One is apt to feel that, in spite of the variety, the command syntax is quite liberal.

Consider the command lines:

```
spell −bv memo
ls −l −t /usr
cat −su junk
tp −mt4 silver
```

Each of these lines fails because of invalid syntax, although the option or key characters are all valid.

We see no reason to continue burdening users with an inconsistent command syntax. "Historical" or "implementation" reasons are weak excuses.

Proposal

We propose that specific command-syntax guidelines be adopted and applied to the user commands in Section 1 of the *UNIX/TS User's Manual.* These guidelines would describe a command syntax that would be accepted by essentially all commands. A few commands (for example, *sort)* may require a syntax outside the scope of these guidelines. The primary goal of this effort is not to reinvent the UNIX command syntax, but to reduce inconsistencies among commands.

Attachment A summarizes the proposed command syntax and offers some examples.

Valid "historical" conflicts should be addressed. Commands may accept a "historical" syntax as well as a more consistent one. However, we see no need to encourage "historical" command syntaxes by recommending them in the *UNIX/TS User's Manual.* Therefore we propose that command syntaxes given in Section 1 of the *UNIX/TS User's Manual* be revised to reflect the desired syntax in a consistent format.

Implementation

A routine to implement the proposed command syntax has been written. This routine, called *optget,* is designed to scan command lines for valid options, return option characters and pointers to option arguments, and issue error messages when syntax violations are detected.

Attachment B contains the *optget* manual page, source listing, and an example based on the *ptx* command.

Acknowledgements

The authors are especially grateful to D. M. Ritchie and R. C. Haight for their critical review and helpful suggestions.

A. S. Cohen

S. B. Olsson

G. C. Vogel

MH-2524-asc/sbo/gcv-troff

Atts.
Attachment A
Command Syntax Summary
Attachment B
OPTGET(3) Manual Page
Continued next page

WITNESSED AND UNDERSTOOD
By ..... Date .....

Atts. cont'd
optget.c — source listing
Example — ptx command

Copy (with att.) to
All members of Department 2524
Laboratory 252 Supervision
USS mailing list
S. R. Bourne
D. S. DeJager
A. G. Fraser
C. B. Haley
S. C. Johnson
B. W. Kernighan
M. D. McIlroy
N-P. Nelson
P. H. Rank
D. M. Ritchie
L. Rosler
K. Thompson

Command Syntax - Summary

The format of a command is as follows:

command:	*name [option(s)] [file(s)]*
name:	The name of an executable file.
option:	'–'*noargletter(s)* or '–'*argletter[ ]optarg* where [ ] is optional white space.
noargletter:	A single letter representing an option without an argument.
argletter:	A single letter representing an option requiring an argument.
optarg:	Argument (character string) satisfying preceding argletter.
file:	File name (or other command argument) not beginning with '–', or '–' by itself indicating the standard input.

Accordingly (ref. *UNIX/TS User's Manual Edition 1.0),* the following are legal and perform the same operation:

```
ptx −ft −w 70  infile outfile
ptx −f −t −w 70 infile outfile
ptx −ft −w70 infile outfile
ptx −f −w70 −t infile outfile
ptx −f −w 70 −t infile outfile
```

while these are *not* legal:

```
ptx −ftw 70 infile outfile
ptx −fw70t infile outfile
ptx −f −w −t infile outfile
```

NAME

optget – get option letter from argv

SYNOPSIS

```
int optget (argc, argv, optstring)
int argc;
char **argv;
char *optstring;
extern char *optarg;
extern int optind;
```

DESCRIPTION

Optget returns the next option letter in *argv* that matches a letter in *opstring*. *Opstring* is a string of recognized option letters; if a letter is followed by a colon, the option is expected to have an argument which may or may not be separated from it by white space. *Optarg* is set to point to the start of the option argument on return from *optget*.

Optget places in *optind* the *argv* index of the next argument to be processed. When all options have been processed (i.e., up to the first non-option argument), *optget* returns EOF.

DIAGNOSTICS

Optget prints an error message on *stderr* and returns a question mark (?) when it encounters an option letter not included in *optstring*.

EXAMPLE

The following code fragment shows how one might process the arguments for a command that can take the mutually exclusive options **a** and **b**, and the options **f** and **o**, which require arguments.

```
main(argc,argv)
char **argv;
{
        int c;
        extern int optind;
        extern char *optarg;
        .
        .
        .
        while((c=optget(argc, argv, "abf:o:")) != EOF)
                switch(c) {
                case 'a':
                        if(bflg) errflg++;
                        else aflg++;
                        break;
                case 'b':
                        if(aflg) errflg++;
                        else bproc();
                        break;
                case 'f':
                        ifile = optarg;
                        break;
                case 'o':
                        ofile = optarg;
                        bufsiza = 512;
                        break;
                case '?':
                        errflg++;
                }
```

```
        if(errflg) {
                fprintf(stderr,"usage: . . . ");
                exit(2);
        }
        if(access(argv[optind], 4)) {
        :
        :
}
```

Feb 2 13:26 1979 optget.c -- argument decoder Page 1

```
#include    <stdio.h>

#define     ERR(s, n, c) fprintf(stderr, s, n, c)

int     optind;
char    *optarg;
char    *strchr();

optget(argc, argv, opts)
char    **argv, *opts;
{

        static sp = 1;
        char c;
        char *cp;

        if(optind == 0) optind++;

        if((sp == 1) && ((optind >= argc) |
          (argv[optind][0] != '-') | (argv[optind][1] == '\0')))
                return(EOF);
        c = argv[optind][sp];
        if((c == ':') | ((cp=strchr(opts, c)) == NULL)) {
                ERR("%s: illegal option -- %c\n", argv[0], c);
                if(argv[optind][++sp] == '\0') {
                        optind++;
                        sp = 1;
                }
                return('?');
        }
        if(*++cp == ':')
                if(sp != 1) {
                        ERR("%s: can't group options with arguments -- %c\n", argv[0], c);
                        sp = 1;
                        optind++;
                        return('?');
                }
                else if(argv[optind][2] != '\0')
                        optarg = &argv[optind++][2];
                else if(++optind >= argc) {
                        ERR("%s: argument required -- %c\n", argv[0], c);
                        sp = 1;
                        return('?');
                }
                else optarg = argv[optind++];
        else if(argv[optind][++sp] == '\0') {
                sp = 1;
                optind++;
        }

        return(c);
}
```

```
main(argc,argv)
char **argv;
{
        int     c;
        extern int optind;
        extern char *optarg;
        .
        .
        while((c = optget(argc, argv, "ftrw:g:i:o:b:")) != EOF) {
                switch (c){
                case 'f':
                        foldf++;
                        break;
                case 't':
                        if(wlen == 0)
                                llen = 100;
                        break;
                case 'r':
                        rflag++;
                        break;
                case 'w':
                        getwlen(optarg);
                        break;
                case 'g':
                        gap = gutter = atoi(optarg);
                        break;
                case 'i':
                        if(only){
                                fprintf(stderr, "Only file already given\n");
                                exit(1);
                        }
                        ignore++;
                        xfile = optarg;
                        break;
                case 'o':
                        if(ignore){
                                fprintf(stderr, "Ignore file already given\n");
                                exit(1);
                        }
                        only++;
                        xfile = optarg;
                        break;
                case 'b':
                        bfile = optarg;
                        break;
                case '?':
                        errflg++;
                        break;
                }
        }
        if(errflg) {
                fprintf(stderr, "usage: ptx [-ftr] [-w width] [-g gap] [-o only]");
                fprintf(stderr," [-i ignore] [-b break] [input [output]]\n");
                exit(1);
        }
        .
        .
}
```