MicroTools

UNIX-Like Utilities for CP/M

MicroToo1s Software
PO Box 12
Napervi11e, IL 60540

January 1983

COPYRIGHT NOTICE

TRADEMARKS

The following trademarks are referenced in this publication:

CP/M	Digital Research, Inc.
UNIX	Bell Laboratories
Wordstar	MicroPro International, Inc.
MicroShell	New Generation Systems, Inc.

CONTENTS

Overview

MicroTools are UNIX-like utilities for CP/M that provide features absent in similar programs from other sources. For example, many listing programs lack a means for offsetting the output from the left margin to allow room for punching holes (PIP also lacks this option). The MicroTool **pr** provides this option and others seldom found in listing programs. The MicroTools employ syntax similar to that of the UNIX operating system. They have logical option defaults, and support input/output redirection, which allows a program's input and output to be specified on the command line that invokes the program.

MicroTools that can operate on a list of files (such as pr and wc) accept wildcards (ambiguous file references) similar to CP/M wildcards. The MicroTool wildcards, however, also allow exclusion of files.

The MicroTools are compatible with New Generation Systems' MicroShell (New Generation Systems, Inc., 2153 Golf Course Drive, Reston, VA 22091). Although not required for operation of the MicroTools, MicroShell is recommended for the following reasons:

- It supports multiple CP/M commands per line, input/output redirection, and pipelines.
- It contains an automatic disk-search mechanism that makes most disk-drive prefixes unnecessary and allows the MicroTools to be placed only in user area 0 yet be accessed from all user areas.
- It contains a command file mechanism that is significantly more powerful than the CP/M SUBMIT program.

- It allows lowercase command-line arguments to be passed to programs.

The MicroTools require CP/M 2.2 with 32K of main memory. They are most effective in environments that have enough mass storage so that they may be kept online at all times, although such an environment is not mandatory.

The documentation supplied with the MicroTools consists of a general information section that describes command syntax, options, wildcards, input/output redirection, and pipelines, and an individual description for each MicroTool. The individual descriptions contain examples to guide the user and to suggest possible applications.

Command Syntax

Application programs can be designed with either of two user-interface philosophies: (i) a verbose, menu-driven philosophy, or (ii) a terse, single-command-line philosophy. Both have advantages and disadvantages. Advocates of the first philosophy feel that users should not need to learn command syntax or consult a manual to use a program; hence, programs should be menu-driven. Advocates of the second philosophy counter that humans are intelligent creatures that learn quickly; therefore, syntax and often-used options are soon committed to memory, after which terseness becomes a valuable attribute. These are human-factors concerns that can be argued almost without end. There is a practical consideration, however, that tends to shift the balance in favor of the second philosophy (at least for the kind of task for which the MicroTools are intended): the need to interconnect general-purpose programs to produce more specific tools.

In discussing UNIX (Datamation, November 1981), Michael Lesk of Bell Laboratories writes:

> UNIX is undoubtedly near an extreme of terseness, partly because it was originally designed for slow hardcopy terminals. However, the terseness is very valuable when connecting processes (programs). If the command that lists the logged-on users prints a heading above the list, you can't tell how many users are on by feeding the command output to a line counter. If the editor types acknowledgements now and then, its output may not be directly usable as input somewhere else. Of course, you could feed it through something which strips off the extra remarks, but presumably that program would add its own chatty messages.

Another point to consider is that the interactiveness of menu-driven programs can be a burden when interconnecting programs. Suppose a user creates a pipeline (pipelines are described in the in next section), in which the output of program 1 is the input to program 2, whose output, in turn, is the input to program 3. In UNIX, this is denoted as follows:

```
program 1 | program 2 | program 3
```

In practice, the user wants to regard the pipeline as a new command and so will probably use a command file or alias to invoke the pipeline with a short mnemonic. The user will want to be able to invoke the pipeline and have it run to completion without stopping at each intermediate program to prompt for inputs. Therefore, the syntax must be based on the terse, single-command-line philosophy.

Because the MicroTools are general-purpose programs designed to be interconnected (either via MicroShell's pipeline mechanism or via the MicroTool **p**, a pipeline processor), the terse, single-command-line philosophy is used. The syntax is sufficiently straightforward that unsophisticated users can master it in a short time. Users familiar with UNIX will be able to use the MicroTools immediately.

General Command Format

The general format for the MicroTools (with some exceptions as detailed in the individual command descriptions) is the command mnemonic followed by options, file-specification, and redirection sections, as follows (brackets indicate that the sections are optional--the brackets are not part of the command line):

```
command [options] file(s) [redirection]
```

For example, the following command prints double-spaced listings of the files text1 and text2 and saves the output in a file named log:

```
pr -d text1 text2 >log
```

The command mnemonic is "pr" (for "print"); the options section is "-d"; the file-specification section is "text1 text2"; the redirection section is ">log".

The options and redirection sections are optional and can be omitted. If the options section is omitted, the program's default options are used (option defaults are given in the individual command descriptions). If the redirection section is omitted, the program takes its input from the named file(s) and sends its output to the named output file (or to the console if an output file is not specified).

The options, file-specification, and redirection sections each have their own syntax, as described below. Common to all three, however, is the concept of an "argument." An argument is a string of characters delimited by blanks (spaces or tabs). For example, the following command consists of six arguments:

```
col -x -q -c4 text >output
```

The arguments are "Col", "-x", "-q", "-c4", "text", and ">output". The concept of an argument is used in the syntax descriptions below and in the individual command descriptions.

Note that a brief help message can be obtained for each MicroTool by simply typing the command name without any following arguments.

Options

The options section is used to select command options, which are detailed in the individual command descriptions. An option is selected via one or more arguments, the first of which always begins with a character followed by a single letter that identifies the option. For example, the following command contains a single-argument option (-c5) that instructs **col** to print the file named infile in five-column format:

```
col -c5 infile
```

The following command contains a two-argument option (-h hello) that instructs pr to print "hello" as the header string:

```
pr -h hello infile
```

To specify more than one option, simply list them; for example, the following command prints infile double-spaced (-d), with numbered lines (-n), with a page length of 55 (-155), and with the header string "goodbye" (-h goodbye):

```
pr -d -n -155 -h goodbye infi1e
```

A useful shorthand is provided that allows sing1eargument options to be concatenated into a single argument. For example, the following two commands are identical:

```
pr -d -n -155 -f infile
pr -dn155f infile
```

A multi-argument option can also be concatenated, but further options must start with a - character. Thus, the following command is leqal:

```
pr -dn155fh goodbye -08m366 infile
```

while the following command is illeqal:

```
pr -dn155fh goodbye 08m366 infile
```

Invalid option identifiers are not accepted and an appropriate error message is printed. Some mistakes, however, are detected but are reported inappropriately. For example, consider the following command:

```
pr -h infile
```

The user omitted the second argument of the -h option and pr assumes that infile is the required argument. Therefore, an input file has not been specified so pr prints a "no input file" error message.

File Specification

The file-specification section defines a command's input and output. Files are specified by their directory file names; explicit drive designations can be used. The MicroTools allow direct input to a command from the console by specifying the file con:. This is, however, a questionable procedure because the input is lost. It is generally preferable that input be contained in files. When inputting directly from the console,

terminate input with a control-z (CP/M end of file) on a line of its own followed by a carriage return.

There are two types of file-specification section (the particular type used by a command is given in the USAGE section of its individual description): (i) infile-outfile, and (ii) list. The infile-outfile type consists of one or two arguments that specify the input file or console (con:) and optional output file or device (con: or 1st:) as follows (brackets indicate optional arguments--the brackets are not part of the command line):

```
command infile [outfile]
```

If an output file or device is not specified, output is sent to the console. An example of the infile-outfile specification type follows:

```
col list table
```

This command prints the file list in multicolumn format and saves the result in the file table.

The list file-specification type consists of one or more arguments that specify input files, as follows:

```
command infile1 infile2 infile3 ...(etc)
```

An output file cannot be specified in a list specification; the output is always sent to console. To collect the output in a file, output redirection must be used, as follows:

```
command infile1 infile2 infile3 >outfile
```

For both types of file-specification section, an appropriate error message is printed if a specified input file cannot be opened. If a specified output file is already present, it is overwritten.

Commands that accept a list specification also accept wildcards (ambiguous file references) similar to CP/M wildcards. The MicroTool wildcards, however, also allow exclusion of files via a @ prefix. For example, the following command prints all files except .com files:

```
pr *.* @*.com
```

Note that if the first wildcard argument is an exclusion wildcard, *.* is assumed; therefore, *.* is superfluous in the example above.

In @ specifiers, the ? character never matches a space. Thus, for example, "@help?" does not match "help". The normal CP/M ? character matches a space at the end of a file name or extension; thus, "help?" matches "help".

If a wildcard is given that does not match any files, a "no input file" error message is printed.

When using **pr** to print a batch of files specified by wildcards, it is useful to be able to verify that the wildcard expansion is as desired before beginning printing. This can be done with the **-w** option of **find**. For example, the following command lists all files that have a .man extension:

```
find -w *.man
```

Redirection

When the MicroTools are used under MicroShell, both input and output redirection are supported. For example, the following command causes pr to take its input from infile and send its output to outfile:

```
pr <infile _outfile
```

Input redirection is invoked by a < character followed by a file name. Either one or two arguments may be used; thus, the following two commands are equivalent:

```
pr <infile
pr < infile
```

When the MicroTools are used at CP/M prompt level (not under MicroShell), the < operator is not supported (the reason for this is highly technical). This is not, however, a serious limitation; because an input file specification is always accepted, the < operator is superfluous, that is, the following commands are identical:

```
pr infile
pr <infile
```

Output redirection is invoked by a > character followed by a file name. Either one or two arguments can be used; thus, the following two commands are equivalent:

```
pr infile >outfile
pr infile >outfile
```

Output redirection is used primarily when the output of a command that operates on a list of input files must be saved in a file or sent to the list device (the file-specification section of list-type commands does not accept an output file specification).

For example, a user may want to generate listings of all .c files and save them in one file for archival purposes. The following command might be used:

```
pr *.c >archive
```

When the MicroTools are used at CP/M prompt level, output is redirected to the list device as follows:

```
prog file >lst:
```

When the MicroTools are used under MicroShell, output is redirected to the list device as follows (although MicroShell can be patched to accept user-defined syntax):

```
prog file >$p
```

It is nonsense to specify both an output file and output redirection.

For input redirection, an appropriate error message is printed if a specified file cannot be opened. For output redirection, if a specified file already exists, it is overwritten.

Pipe1ines

A pipeline is a mechanism for interconnecting programs. The usual notation is as follows:

```
programl | program2 | program3
```

This command connects the output of programl to the input of program2 and connects the output of program2 to the input of program3. The value of a pipeline mechanism is succinctly stated by Kernighan and Plauger in Software Tools:

> A consideration in favor of the pipeline is that it encourages the construction of smaller programs to do simpler functions. These smaller programs are much easier to write, debug, document, maintain, and improve independently than they would be if combined into a single monster. And of course separate programs can be combined in novel ways, something which is hardly possible if they have already been combined in some "obvious" way.... Many jobs will not get done at all unless they can be done quickly. Efficiency is hardly of importance for a temporary hookup meant to be used only a few times. Should a particular organization of tools prove so useful that it begins to consume significant resources, then you can consider replacing it with a more efficient version.

Pipelines are ideally implemented by executing the programs as concurrent processes. CP/M lacks resources to support concurrent processes. However, pipelines can be simulated under CP/M by using temporary files to communicate between

programs. For example, the pipeline "programl | program2 | program3" can be simulated as follows:

```
program | >temp0
program2 <temp0 >temp1
program3 <temp1
era tempo
era tempI
```

MicroShell supports pipelines directly and transparently through use of its input/output redirection mechanism and temporary files as shown above. At CP/M prompt level, however, the **<** operator is not supported. The simulated pipeline will run correctly if the **<** characters are removed (recall that an input file is always accepted).

As can be seen from the above example, the simulated pipeline is not as clean and direct as the | pipeline notation; considerable work is required to manually convert a pipeline to its simulated form and the conversion is error-prone (the temporary files must be correctly aligned). To remove this burden from the MicroTools user working at CP/M prompt level, a pipeline-processor, **p**, is included with the MicroTools. **p** converts a pipeline specification given in standard notation to its simulated form, saves the results in a file, and executes the file via CP/M's SUBMIT program. **p** is fully described in its individual command description. (**p** is not necessary under MicroShell.)

MicroShell Compatibility

New Generation Systems' MicroShell is a command interpreter that adds UNIX-like features to CP/M (some of these features are listed in the MicroTools Overview). MicroShell loads as a transient command (.com file) and relocates itself to high memory, overwriting the console command processor (CCP) and replacing and extending its functions. Because MicroShell occupies about 8K bytes of memory at the top of the transient program area, the available memory must be 32K + 8K = 40K.

Required settings for MicroShell are as follows:

```
G = off   (to disable MicroShell's line-feed
           gobbling)
M = off   (for UNIX mode)
T = off   (to let MicroShell process special
           characters)
D 4 = 0   (to disable the input ignore feature)
```

In addition, the following settings are recommended:

```
F = on    (to enable file search)
U = off   (to allow lowercase letters to be
           passed via the command line)
V = off   (to disable command echo)
```

The MicroShell syntax for output redirection to the list device is:

```
command >$p
```

This differs from the syntax for MicroTools at the CP/M prompt level (not under MicroShell):

```
command >lst:
```

As an alternative to >$p, MicroShell can be configured to accept >lst:. The remainder of this manual assumes that this reconfiguration has been made.

The MicroTools cannot properly handle double-quoted argument strings under MicroShell because double-quote characters (") are MicroShell special characters. To avoid this problem, either use single quotes or escape the double quotes to allow the MicroTools' intrinsic argument processing to handle the double quotes, as shown in the following example:

```
pr -h \"user header" infile
```

In either case, MicroShell parses extra blank space between arguments to a single space. This means multiple spaces cannot be passed to programs on the command line.

Direct console input does not work properly when output is redirected (for example, "pr -t con: >file") due to a conflict between the MicroShell and MicroTool redirection mechanisms. To achieve the desired effect, backslash the> character to allow the MicroTool to handle the redirection, as follows:

```
prog                    con:                    \>file
```

Command Descriptions

This section contains individual command descriptions for the MicroTools. The description for each MicroTool consists of a few pages that adhere to a standard format:

PROGRAM	Gives the command name.
USAGE	Summarizes the command's syntax.
DESCRIPTION	Describes the function of the command and its options.
EXAMPLES	Gives typical applications and examples to illustrate fine points.
CAVEATS	Lists possible pitfalls to avoid.

The command descriptions are arranged alphabetically by command name. There is no overall page numbering for this section to allow insertion of additional command descriptions.

PROGRAM

col -- print file in multicolumn format

USAGE

col [options] infile [outfile] [redirection]

DESCRIPTION

col prints the named infile to the named outfile. If outfile is not given, output is sent to the console. By default, the output is in four-column format with vertical scanning and a page length of 20. Typing control-c aborts printing.

Various options are available to alter the output format. Options may appear singly or combined in any order. The options are as follows (substitute desired numbers for characters not in boldface):

-a Print multicolumn across the page (horizontal scanning). This option works by setting the page length to 1. Therefore, do not reset the page length via the -1 option when using this option. The -a option is not compatible with the -r option.

-cn Produce n columns of output (default is 4).

-en Expand input tabs with spaces to character positions n+1, 2n+1, 3n+l, etc. n may not be omitted or O. By default, tabs are expanded with n equal to 8.

-ln Set page length for vertical-scanning mode (-a option not selected) to n (default is 20).

-pn Pad n blank lines between pages.

-q Do not equalize column lengths on final page.

-rn Enable record mode. The input is treated as multiline records delimited by blank lines. Records are printed in multicolumn format without breaking them across column boundaries. The records can be of different lengths. n should be set to the maximum record length. Undefined results are obtained if the page length is

set to a value less than the maximum record length. This option is not compatible with the -a option.

-s Use the next argument as a column-delimiter string. If the string contains blanks or tabs, it must be enclosed in single or double quotes.

-wn Set the line width to n (default is 80).

-x Add page cut marks.

EXAMPLES

Consider a file named list containing the numbers to 20, with one number per line. The command:

```
col -w40 list
```

produces the following vertically-scanned output:

```
1         6         11        16
2         7         12        17
3         8         13        18
4         9         14        19
5         10        15        20
```

The command:

```
col -w40a list
```

produces the following horizontally-scanned output:

```
1         2         3         4
5         6         7         8
9         10        11        12
13        14        15        16
17        18        19        20
```

Print file in five-column format with a page length of 60:

```
col -c5l60 file
```

Print file in three-column across-the-page format with the string " * " delimiting the columns:

```
col -c3as II " * " file
```

Print file of address records (maximum record length of 5) in 4-column format with a page length of 60:

```
col -160r5 file
```

Print the same file of address records in the same format on the list device, but paginate the output using pr (pr's header and footer space by default consists of 13 lines; therefore, the col page length is set to 13 less than the pr page length [default 66]):

```
col -153r5 file temp
pr temp >lst:
```

or:

```
col -153r5 file | pr >lst:
```

To print a file of address records on label stock, set the page length to the number of lines on a label, set the line width to align the records horizontally on the labels, set the padding between pages to the number of lines (vertically) between labels (if any), use the record mode, and select the appropriate number of columns. A typical example for three-across labels is:

```
col -18w132r6c3 file 1st:
```

CAVEATS

Input lines that are too long to fit within a column are silently truncated.

Because the -a option works by setting the page length to 1, the -x option is not very useful with the -a option.

PROGRAM

cut -- cut character or field columns from a file

USAGE

cut [options] infile [outfile] [redirection]

DESCRIPTION

cut prints specified character columns or field columns. If outfile is not given, output is sent to the console.

Various options are available to alter the behavior of cut. Options may appear singly or combined in any order. The options are as follows (substitute desired values for characters not in boldface):

-cs Cut the character columns specified by the list s. The specification s consists of comma-separated numbers that specify the columns to be cut. For example, "cut -c1, 2, 3 file" cuts columns 1 to 3. A range may be specified instead of a number; thus, "cut -c5,8,10 20,50 file" cuts columns 5, 8, 10 through 20, and 50. By default, cut columns are flushed to the left.

-dc Take the delimiter for the -f option to be the character c (default delimiter is the : character).

-fs Cut the field columns specified by the list s. A field is defined as the characters between two delimiter characters. The nth field is the field beginning after the nth delimiter character. When a field is cut, the leading delimiter character is retained. The syntax for s is identical to that described for the **-c** option except that the numbers refer to fields rather than columns. Cut fields are always flushed to the left.

-r Retain original column positions under the **–c** option (do not flush left).

-x Cut columns or fields not specified.

EXAMPLES

File test is as follows:

```
123456789
123456789
abcdefghi
123456789
123456789
```

Cut columns 1 through 3, 7, and 9 from the file test:

```
cut -cl-3,7,9 test
```

The output is as follows:

```
12379
12379
abcgi
12379
12379
```

Perform the same function, but retain the original spacing:

```
cut -rcl-3,7,9 test
```

The output is as follows:

```
123   7 9
123   7 9
abc   g i
123   7 9
123   7 9
```

The file people contains the following:

```
:Bonaparte,Napolean:Emperor:France
:Antoinette, Marie:Queen:France
:Franklin, Ben:Scientist:USA
:Thatcher, Margaret:Prime Minister:England
```

Print only the names people and countries from the file people:

```
cut -f1,3 people
```

The output is as follows:

```
:Bonaparte, Napolean:France
:Antoinette, Marie:France
:Franklin, Ben:USA
:Thatcher,Margaret:England
```

PROGRAM

crypt -- encrypt and decrypt files

USAGE

```
crypt [-d] key infile [outfile] [redirection]
```

DESCRIPTION

crypt encrypts and decrypts files based on a variable user-specified key. For encryption, the **-d** option is omitted. decryption, For decryption, the **-d** option is included. If outfile not given, output is sent to the console.

EXAMPLES

Encrypt the file named salaries using the key "insideout" and save the results in a file named secret:

```
crypt insideout salaries secret
era salaries
```

Decrypt the file secret from the previous example:

```
crypt -d insideout secret salaries
```

CAVEATS

The standard caveats relating to key selection and safeguarding apply here. Choose keys that are "unobvious" (there is a tradeoff between difficulty of guessing for a would-be intruder and ease of recall for the user). Do not leave keys lying around. Do not leave any clear text in proximity to its encrypted output.

To make breaking the program a little harder, no details are given here on the algorithm used. It suffices to say that repeated sequences in the clear text will not cause repeated appearances of the transformed key.

If you forget your key, you're in big trouble!

PROGRAM

deform -- deformat a file

USAGE

deform [options] infile [outfile] [redirection]

DESCRIPTION

deform removes text formatter control lines (lines that begin with a special character--by default, a period) from the named infile and sends the results to the named out file. Optionally, deform can also remove in-line commands (two-character sequences beginning with a special character--by default, a \ character) and control-character commands (such as those used by Wordstar). For in-line commands, double backslashes (\\) are retained as one backslash (\). For control-character commands, control-o is mapped to a space and control-underscore (_) is mapped to a hyphen; all other control characters except line feeds, tabs, and carriage returns are discarded. If outfile is not given, output is sent to the console.

Various options are available to alter the behavior of deform. Options may appear singly or combined in any order. The options are as follows (substitute desired characters for characters not in boldface):

-ec Take the key for an in-line command to be the character c instead of a backslash (,). Two character sequences beginning with the character c are removed. Double c characters are retained as one c character. Nonpaddable spaces (character c followed by a space) are retained as spaces.

-fc Take the key for a control line to be the character c instead of a period. All lines with the character c in the first position are removed.

-i Enable in-line command stripping; backslashes (or alternative specified characters) get special treatment as described above.

-n Under the -wand -x options, retain numbers as valid word components (for example, "test55" is considered to be a word).

-r Reverse the behavior of deform (retain control lines in the output and discard noncontrol lines).

-s Enable control-character command stripping; control characters get special treatment as described above. (For use with Wordstar files.)

-w Generate a list of the words in the file, with one word per line. A "word" is a string of characters containing only letters or apostrophes. Uppercase letters are mapped to lowercase.

-x Same as the -y option but uppercase letters are not mapped to lowercase.

EXAMPLES

Remove control lines and in-line commands from the file named text and save the results in a file named clean:

```
deform text clean
```

Remove all lines from the file text that contain a : character in the first position:

```
deform -f: text
```

Generate a list of the words used in the file text:

```
deform -w text
```

Print all lines in the file text that begin with character the character *:

```
deform -rf* text
```

Count the total number of different words used in the file text and print the result:

```
deform -w text | sort | uniq | wc -1
```

PROGRAM

diff -- compare text files

USAGE

diff file1 file2

DESCRIPTION

diff performs a line-by-line comparison of two text files and reports any differences found. High bits are stripped from all files for Wordstar compatibility. The difference report consists of lines of the following form:

n1an2,n3 (or n1,n2an3)
n1,n2dn3 (or n1dn2,n3)
n1,n2cn3,n4

The letter a denotes text that has been appended; d denotes deleted; c denotes changed. Numbers (n1...n4) before the letter (a, d, or c) refer to file1; those after the letter refer to file2. For example, 17a8,12" indicates that 5 lines (8 through 12 in file2) are appended after line 7 in file1. After each line of this form, the affected lines in file1 are printed flagged with a < character followed by the affected lines in file2 flagged by a > character.

EXAMPLES

Compare the files named old and .new:

diff old new

Under MicroShell, the difference report can be saved as follows:

diff old new >report

PROGRAM

echo -- echo arguments

USAGE

echo [options] arguments [redirection]

DESCRIPTION

echo prints its arguments on the console, terminating them with a CR-LF. If the **-n** option is included, the CR-LF is suppressed.

EXAMPLES

Print the message "compilation complete" by including the following command within a submit file:

```
echo compilation complete
```

Suppose a user wishes to keep a log of the number of times a particular submit file has been executed. The following command is placed in the submit file:

```
echo -n + >log
```

To get a count, the user types:

```
wc -c log
```

Create a one-line file without loading the editor:

```
echo This is a test file. >test
```

CAVEATS

Under MicroShell, the %print command is available, so echo is not necessary.

PROGRAM

find -- find pattern in files

USAGE

find [options] file_list [redirection]

DESCRIPTION

If the **-p** option is not selected, find prompts for a pattern (character string), searches for the pattern in the specified files, and prints the lines containing the pattern on the console. Wildcard file references are accepted. If more than one file is specified, or if wildcards are specified, find identifies the input files in which the pattern occurrences are found. If the **-p** option is selected, the pattern is taken from the command line (as described below) and find does not prompt for a pattern.

Various options are available to alter the behavior of find. Options may appear singly or combined in any order. The options are as follows:

-a Include patterns split across line boundaries (raw mode). For example, the pattern 110123456" matches the following two lines:

```
The first seven numbers are 0123
456.
```

When a split occurrence is found, both lines are printed. The **-a** option is not compatible with the **-x** option.

-b Include patterns split across line boundaries (text mode). For example, the pattern limy friendly cat" matches the following two lines:

```
Please say hello to my friendly cat.
```

When a split occurrence is found, both lines are printed. "Words" (that is, character strings not containing blanks, tabs, or newlines) that are split across a line boundary must be hyphenated in the input file.

If a pattern is split after a period, the pattern specification must contain two spaces after the period. If a pattern is split after any other punctuation mark (except a hyphen), the specified pattern must contain one space after the punctuation mark. The **-b** option is not compatible with the **-x** option.

-i Enable identifier mode. An occurrence of a pattern in an input file is found only if the character immediately before the occurrence and the character immediately after the occurrence are not letters, numbers, or the underscore character. Thus, for example, the pattern "cat" would not be found in the string "vacate" but would be found in the string limy cat is good."

-n Precede each output line with its line number in the input file.

-p Take the next argument to be the search pattern. All letters (whether given as uppercase or lowercase) are taken to be lowercase; to specify a letter as uppercase, precede it with a ~ character. If the pattern contains blanks or tabs, it must be enclosed in single or double quotes. This option is provided for using find with output redirection and within pipelines.

-w Verify wildcard expansions. The names of files that match the wildcard specifications given on the command line are printed. Pattern searching is not performed.

-x Print all lines except those containing the specified pattern.

EXAMPLES

Find occurrences of the pattern "cat" in the file named text (find's prompt is in italics):

```
find text
find what pattern? cat
```

Find occurrences of the identifier "line" in all .c (C-language) files and give the line number of each occurrence:

```
find -ni *.c
find what pattern? line
```

Find occurrences of the pattern "Yes, master" in the file text, including occurrences split across line boundaries:

```
find -b text
find what pattern? Yes, master
```

Print lines in the file text that do not contain the string "REM":

```
find -x text
find what pattern? REM
```

Verify expansion of the wildcard "text??man":

```
find -w text??man
```

Print all words the in file text that contain the pattern "ing":

```
deform -w text | sort | uniq | find -p ing
```

CAVEATS

The dual pattern-specification mechanism used by find is made necessary by the fact that CP/M always maps lowercase command lines to uppercase. The recommended usage for find is to omit the -p option and let find prompt for the pattern. In this case, the pattern can be typed directly in uppercase and lowercase without delimiting the pattern with single or double quotes. However, when the output of find is redirected, the pattern prompt is also redirected. The -p option should be used in this case. The-p option is also required when find is used in a pipeline.

The *, ?, and @ characters cannot be specified in arguments other than wildcard file references because they confuse the wildcard expansion code.

Note that the MicroTool wildcards differ slightly from standard CP/M wildcards; refer to the Command Syntax description for details.

PROGRAM

p -- implement pipeline at CP/M prompt level

USAGE

p [options] pipeline_specification

DESCRIPTION

p converts a UNIX-like pipeline specification to a file called pype.sub that simulates the pipeline and then automatically invokes the CP/M SUBMIT program (which must reside on drive a:) to execute the pipeline. Either a | or a ! character can be used to denote a pipe. Pipeline output is sent to the console unless overridden by the **-o** option. **p** may reside on any drive and pipelines can be executed while logged onto any disk. Redirection is not allowed.

Various options are available to alter the behavior of p. Options may appear singly or combined in any order. The options are as follows:

-o Save the final output from the pipeline in the file given by the next argument. The list device (lst:) may be specified.

-s Save the generated pipeline simulation in the file given by the next argument. This option automatically invokes the **-x** option.

-x Do not execute the generated .sub file.

EXAMPLES

Print file in four-column format and paginate the results:

```
P col -153 file | pr
```

Perform the same function but send the output to the list device:

```
p -o 1st: col -153 file | pr
```

Count the total number of different words used in the file named text and print the result:

```
p deform -w text | sort | uniq | wc -1
```

Generate a word-frequency table sorted by words for the file text (these tables can be used to detect overused words and variant spellings):

```
p deform -w text | sort | uniq -n | col
```

Perform the same function as the above example but sort by count:

```
p deform -w text | sort | uniq -n | sort -nr | col
```

Pipelines can include parameter substitution. Consider the following command:

```
p -s a:column.sub col -153 $1 | pr
```

This saves a .sub file on drive a: named column. sub that can be used to produce a four-column paginated printout of a file, as follows:

```
submit column file
```

Digital Research, Inc. will provide upon request a patch to SUBMIT that allows nested submit files. The patched SUBMIT allows p-generated .sub files to be invoked within other p-generated .sub files.

CAVEATS

The programs used within p pipelines must support output redirection (as do the MicroTools).

Because MicroShell provides a built-in pipeline mechanism, p is not necessary (and, in fact, does not work) under MicroShell.

PROGRAM

paste -- concatenate files horizontally

USAGE

paste [options] file_list [redirection}

DESCRIPTION

paste concatenates the named files (at least two must be specified) horizontally in the order in which the files are specified. Up to six files may be specified. To input from the console at an arbitrary point in the order, use the explicit device name con:.

Various options are available to alter the behavior of paste. Options may appear singly or combined in any order. The options are as follows (substitute desired numbers for characters not in boldface):

-cn Paste each file in a separate column using a column width of n. Lines that are too long to fit in a column are silently truncated.

-d Take the next argument as the delimiter string between files. If the string contains blanks or tabs, it must be enclosed in single or double quotes. The default delimiter string is a single tab character.

-p Paste a prefix. The next argument is pasted onto the beginning of each output line. If only one file is specified, an error message will not be printed. If the prefix string contains blanks or tabs, it must be enclosed in single or double quotes.

-s Paste a suffix. The next argument is pasted onto the end of each output line. If only one file is specified, an error message will not be printed. If the prefix string contains blanks or tabs, it must be enclosed in single or double quotes.

EXAMPLES

File a is as follows:

```
a
aaa
a
aaaaaaaaaa a
aa
a
aaa
```

File b is as follows:

```
bbb
b
bbb
bbbbb
b
b
bb
b
bbb
b
```

Paste files a and b using the default tab delimiter:

```
paste a b
```

The output of the above command is as follows:

```
a       bbb
aaa     b
a       bbb
aaaaaaaaaa      bbbbb
a       b
aa      b
a       bb
aaa     b
bbb
b
```

Paste files a and b in 15-character-wide columns and save the results in a file called save:

```
paste –c15 a b >save
```

The file save is now as follows:

```
a               bbb
aaa             b
a               bbb
aaaaaaaaaa      bbbbb
a               b
aa              b
a               bb
aaa             b
                bbb
                b
```

Paste the prefix "=+=" onto file a:

```
paste -p =+= a
```

The output is as follows:

```
=+=a
=+=aaa
=+=a
=+=aaaaaaaaaa
=+=a
=+=aa
=+=a
=+=aaa
```

CAVEATS

Tabs in the input files disturb columniation with the -c option. Use "pr -t file" as a filter to expand tabs.

paste does not accept wildcard file specification because the file list resulting from wildcard expansion is in an arbitrary order.

PROGRAM

pr -- print files

USAGE

pr [options] file_list [redirection]

DESCRIPTION

Pr prints the named files on the console. Wildcard file references are accepted. By default, the listing is paginated with a header at the top of each page containing the file name and page number. Typing control-c aborts the listing.

Various options are available to alter the appearance of the listing. Options may appear singly or combined in any order. The options are as follows (substitute desired characters or numbers, as appropriate, for characters not in boldface):

-bc Enable break processing. When the specified character c is encountered in the first position of a line, pr breaks to the next page without printing the line. By default, break processing is disabled.

-cn Produce n copies of the output. By default, n is 1.

-d Double space the output.

-en Expand input tabs with spaces to character positions n+1, 2n+1, 3n+1, etc. If n is omitted or 0, input tabs are not expanded. By default, tabs are expanded with n equal to 8.

-f Use form-feed characters to move to new pages instead of a sequence of CR-LF's. This option is useful when output is sent to the console; 'after a partial last page is printed, the output does not scroll off the screen.

-g "Display on glass." This option is equivalent to setting the following options individually: -t -p -123.

January 1983

-h Use the next argument as the header to be printed instead of the file name. If the header string contains blanks or tabs, it must be enclosed in single or double quotes.

-i Use the next argument as a secondary header to be printed to the right of the file name or user-specified header (normally used as a date field). If the secondary header string contains blanks or tabs, it must be enclosed in single or double quotes.

-kn Begin page numbering at n (default is 1).

-ln Set page length to n lines (default is 66).

-mxyz Set the space above the header to x lines; the space below header to y lines; space at bottom of page to z lines. x, y, and z must all be in the range 0 to 9. The defaults are x = 3, y = 3, and z = 7.

-n Add 5-digit line numbers to the beginning of each line. Two spaces separate the number from the beginning of the line.

-on Offset each line n character positions from the left margin.

-p Pause between pages. pr waits for any character to be typed after printing each page. This option does not work when input redirection is in effect.

-s Suppress the header line (but do not suppress pagination).

-t Suppress pagination.

-u Map output to uppercase.

-x Send the following decimal arguments to the output device before printing. This option is useful for sending printer control codes to the list device. If there are no arguments after the final decimal argument, pr terminates immediately.

EXAMPLES

Print file without pagination (equivalent to CP/M's "type file"):

```
pr -t file
```

Concatenate filel and file2 to produce file3 (equivalent to CP/M's "pip file3=filel,file2"):

```
pr -t filel file2 >file3
```

Produce paginated hard-copy listings of filel and file2 with line numbering, lO-character offset from the left margin, and the date (8-5-82) in the header:

```
pr –nol0i 8-5-82 filel file2 >lst:
```

Perform the same function as the previous example but print all .c files and send the decimal code 6 to the list device before printing:

```
pr –nol0i 8-5-82 -x 6 *.c >lst:
```

Print file on a printer that accepts only single sheets; the pause option is used to give time to feed the sheets and the space above the header is set to zero so that the paper may be inserted with the desired header position under the printhead (it is difficult to position single sheets so that the top of the page is under the printhead):

```
pr -pmO37 file >lst:
```

CAVEATS

The -p and -g options do not work when input redirection is in effect. Because an input file specification can always be used instead of input redirection, this is only significant within pipelines. A temporary output file can be used to achieve the desired effect; for example, use:

```
programl | program2 >temp
pr -g temp
```

instead of:

```
programl | program2 | pr -g
```

The *, ?, and @ characters cannot be specified in arguments other than wildcard file references because they confuse the wildcard expansion code.

Note that the MicroTool wildcards differ slightly from standard CP/M wildcards; refer to the Command Syntax section for details.

PROGRAM

rec -- reformat record lines

USAGE

rec [options] infile [outfile] [redirection]

DESCRIPTION

rec reformats single-line multifield records into multiple-line records. By default, rec performs the following functions: (1) replaces each delimiter character (default delimiter is the: character) by a CR-LF, except for the first delimiter in the file, which is simply deleted, and (2) adds an extra CR-LF after each record. If outfile is not given, output is sent to the console.

Various options are available to alter the behavior of rec. Options may appear singly or combined in any order. The options are as follows (substitute desired characters for those not in boldface):

-dc Take the delimiter to be the character c (default delimiter is the: character).

-n Perform a "name swap" on the first field of each input line; fields of the form "lastname, firstname" are converted to "firstname lastname".

-s Suppress the extra CR-LF between records.

EXAMPLES

Suppose that a mailing list is maintained as a file of multifield lines in the file named list, as follows:

```
:Smith, John:2345 W. Armitage Rd.:Lisle, IL 60532:H8/H17/H19
:Jones, Dorothy:651 E. Main St.:Naperville, IL 60540:TRS-80
:Brown, Leroy:4689 S. First Ave.:Frankfort, IL 60423:Apple II+
```

The first field contains the names arranged with the last name first to allow sorting. The second field contains the street address. The third field contains the city, state, and zip code. The fourth field contains information on the computer system owned. Suppose now that a mailing list must

be generated from these records. The following commands generate a file suitable for input to col for label printing:

```
cut -fl-3 list temp
rec -n temp final
era temp
```

or:

```
cut -fl-3 list | rec -n >final
```

The file final is now as follows:

John Smith
2345 W. Armitage Rd. Lisle, IL 60532

Dorothy Jones
651 E. Main St. Naperville, IL 60540

Leroy Brown
4689 S. First Ave. Frankfort, IL 60423

find can be used as a preprocessor to select only those people who own a particular computer, live in a particular state, etc.

PROGRAM

sort -- sort file

USAGE

sort [options] infile [outfile] [redirection]

DESCRIPTION

sort sorts lines of the named infile and sends the results to the named outfile (infile and outfile may be the same file). If outfile is not given, output is sent to the console. By default, the ordering is lexicographic based on entire lines.

Various options are available to alter the ordering. Options may appear singly or combined in any order. The options are as follows (substitute desired characters or numbers, as appropriate, for characters not in boldface):

-b Ignore leading white space (blanks and tabs) for lexicographic comparisons.

-dc Take the delimiter for the -f option to be the character c.

-fn Enable field mode. The sort keys on the nth field. A field is defined as the characters between two delimiter characters (default delimiter is the : character). The nth field is the field beginning after the nth delimiter character.

-m Map uppercase to lowercase for comparisons (uppercase and lowercase sort together).

-n Sort lines first keying arithmetically on an initial numeric string and then keying lexicographically on the remainder of the line or field. Leading white space before the numeric string is ignored and + and - signs are interpreted.

-r Reverse the order of sorting.

EXAMPLES

Sort a file named list and save the results in a file named newlist:

```
sort list newlist
```

Suppose the file named records contains the following lines:

```
:Bonaparte,Napolean:France
:Franklin, Benjamin:USA
:Livingstone, David:Scotland
:Antoinette, Marie:France
:Thatcher, Margaret:England
:De Gaulle, Charles:France
```

Sort the file records so that the lines are grouped by country of residence, and sorted by name within each group:

```
sort -fl records records
sort -f2 records records
```

The result is:

```
:Thatcher, Margaret:England
:Antoinette, Marie:France
:Bonaparte, Napolean:France
:De Gaulle, Charles:France
:Livingstone; David:Scotland
:Franklin, Benjamin:USA
```

Note that this sort is "stable" (a fancy way of saying that if two lines are judged equal by the comparison code, their prior order is not disturbed). Thus, successive sorts based on different fields give a great deal of flexibility, as shown by the example above.

Generate a word-frequency table sorted by words for the file text (these tables can be used to detect overused words and variant spellings):

```
deform -w text | sort | uniq –n | col
```

Perform the same function as the above example but sort by count:

```
deform -w text | sort | uniq –n | sort -nr | col
```

CAVEATS

sort uses the Shell-sort algorithm and is thus not one of the fastest sorts for large files. On large files, the disk drives may time out while sort is working on memory, giving the appearance that sort has locked up--be patient.

sort loads the entire file into memory; thus, the size of files that can be sorted is limited by available memory. Also, sort contains a pointer array that allows it to sort files containing up to 5000 lines. To sort a file containing more than 5000 lines, the file should be split using spl, each chunk should be sorted individually using sort, and the resulting chunks should be merged. The MicroTool merge is in preparation and will be supplied in an update.

PROGRAM

spl -- split file into chunks

USAGE

spl [options] file

DESCRIPTION

spl splits the named file into chunks and saves each chunk in a file. Options allow chunks to contain a fixed number of either lines or characters; the default behavior is to split the input file into 100line chunks. By default, the chunk files are named x0, xl, x2, etc., but x can be replaced with a user specified name.

Various options are available to alter the behavior of spl. Options may appear singly or combined in any order. The options are as follows (substitute desired numbers for characters not in boldface):

-cn Chunk by characters with n characters per chunk. CR-LF pairs are not split and count as one character.

-In Chunk by lines with n lines per chunk.

-n Use the next argument for chunk names. For example, "-n chunk" generates chunks named chunk0, chunk 1 , chunk2, etc. The name may not contain blanks or tabs and must produce valid CP/M file names.

EXAMPLES

Split the file named text into chunks that each contain 100 lines:

spl text

Split the file named text into chunks that each contain 1000 characters:

spl -c1000 text

Split the file named text into chunks that each contain 50 lines and name the chunks bunch0, bunchl, bunch2, etc.:

```
spl -150n bunch text
```

CAVEATS

The maximum number of lines or characters per chunk is 64000.

PROGRAM

str -- display printable strings

USAGE

str [options] infile [outfile] [redirection]

DESCRIPTION

str searches the specified file (usually a *.com file) for strings containing at least four printable characters with at least one letter and terminated with either a CR-LF, a null (0), or a $ character and prints the strings. If outfile is not given, output is sent to the console. str can be used to identify unknown object files, to print all possible error messages, etc.

Various options are available to alter the behavior of str. Options may appear singly or combined in any order. The options are as follows:

-a Precede each string with the address in hex at which the string begins (assumes a *.com file loaded at 100H).

-o Precede each string with the offset in hex of the string from the beginning of the file.

-r Enable raw mode. A string of 3 or more printable characters is printed without requiring any specific terminator.

EXAMPLES

Find all strings in the file pip.com that are terminated in LF-CR, CR-LF, null, or $ (this prints all of pip.com's error messages):

str pip.com

Find all strings in pip.com regardless of terminator:

str -r pip.com

Perform the same function as the above example, but precede each string with its address and save the results in a file named messages:

```
str -ra pip.com messages
```

CAVEAT$

Some object files store strings without specific terminators. Therefore, use the **-r** option to ensure that significant strings are not overlooked.

The string-finding algorithm is fairly primitive. Too much output is produced rather than too little.

PROGRAM

tee -- save pipeline intermediate results (pipefitting)

USAGE

```
tee file
```

DESCRIPTION

tee saves intermediate results within a pipeline in a named file, but is otherwise transparent to data flow through the pipeline.

EXAMPLES

Generate two word-frequency tables for the file text, one sorted by words (saved in the file wtable) and the other sorted by count (saved in the file ctable):

```
deform -w text | sort | uniq -n | tee temp |
     sort -nr | col >ctable
col temp wtable
```

Note that the pipeline command above is typed on one line (it is shown on two lines only because it won't fit within the page margins).

CAVEATS

tee works only under MicroShell.

PROGRAM

uniq -- filter duplicate lines

USAGE

uniq [options] infile [outfile] [redirection]

DESCRIPTION

uniq reads the named infile and compares adjacent lines. By default, uniq writes unduplicated lines and the first copy of duplicated lines to the named outfile. If outfile is not given, output is sent to the console. Duplicated lines must be adjacent to be detected (use the MicroTool sort to bring separated duplicates together).

Various options are available to alter the behavior of uniq Options may appear singly or combined in any order. The options are as follows:

-d Output only the first copy of duplicated lines; unduplicated lines are suppressed.

-n Precede each output line with a count of the number of occurrences of the line.

-u Output only unduplicated lines; all duplicated lines (including the first copies) are suppressed.

EXAMPLES

Strip duplicate lines from the sorted file named list and save the results in a file named newlist:

uniq list newlist

Report duplicated lines in the sorted file list:

uniq -d list

Generate a word-frequency table sorted by words for the file text (these tables can be used to detect. overused words and variant spellings):

```
deform -w text | sort | uniq -n | col
```

Perform the same function as the above example but sort by count:

```
deform -w text | sort | uniq -n | sort -nr | col
```

CAVEATS

Selecting both the -d option and the -u option is nonsense; all lines are suppressed.

PROGRAM

wc -- count lines, words, and characters

USAGE

wc [options] file_list [redirection]

DESCRIPTION

wc counts lines, words, and characters in the named files, and sends the results to the console. If more than one file is specified, count totals are printed. Wildcard file references are accepted.

Various options are available to alter the behavior of wc. Options may appear singly or combined in any order. The options are as follows (substitute desired characters for characters not in boldface):

-c Count characters only. Carriage returns and line feeds are not included in the count. If this option is combined with the **–l** or **–w** option, the last-selected option determines what is counted.

-f Include text formatter control lines ("dot commands"). By default, wc ignores control lines.

-gc Take the key for a control line to be the character c instead of a period ("dot"). Lines with the character c in the first position are excluded from the counts.

-l Count lines only. If this option is combined with the **-c** or **-w** option, the last-selected option determines what is counted.

-w Count words only. If this option is combined with the **-c** or **-l** option, the last-selected option determines what is counted.

EXAMPLES

Count lines, words,. and characters in the file named text and ignore text processor control lines:

```
wc text
```

Count lines, words, and characters in the files textl, text2, and text3 and include text processor control lines ("dot commands"):

```
wc -f textl text2 text3
```

Count the total number of different words used in the file text and print the result:

```
deform -w text | sort | uniq | wc -l
```

CAVEATS

The * , ? , and @ characters cannot be specified in arguments other than wildcard file references because they confuse the wildcard expansion code.

Note that the MicroTool wildcards differ slightly from standard CP/M wildcards refer to the Command Syntax section for details.

When printing the output on a list device that cannot process tab characters, expand tabs by filtering the output with "pr -t"