COL

NAME

col — filter reverse line feeds from input

SYNOPSIS

col [**-bfpx**] [**-l** *num*]

DESCRIPTION

Col filters out reverse (and half reverse) line feeds so the output is in the correct order with only forward and half forward line feeds, and replaces white-space characters with tabs where possible. This can be useful in processing the output of `nroff`(1) and `tbl`(1).

Col reads from standard input and writes to standard output.

The options are as follows:

-b Do not output any backspaces, printing only the last character written to each column position.

-f Forward half line feeds are permitted ("fine" mode). Normally characters printed on a half line boundary are printed on the following line.

-p Force unknown control sequences to be passed through unchanged. Normally, **col** will filter out any control sequences from the input other than those recognized and interpreted by itself, which are listed below.

-x Output multiple spaces instead of tabs.

-l*num*
Buffer at least *num* lines in memory. By default, 128 lines are buffered.

The control sequences for carriage motion that **col** understands and their decimal values are listed in the following table:

ESC−7	reverse line feed (escape then 7)
ESC−8	half reverse line feed (escape then 8)
ESC−9	half forward line feed (escape then 9)
backspace	moves back one column (8); ignored in the first column
carriage return	(13)
newline	forward line feed (10); also does carriage return
shift in	shift to normal character set (15)
shift out	shift to alternate character set (14)
space	moves forward one column (32)
tab	moves forward to next tab stop (9)
vertical tab	reverse line feed (11)

All unrecognized control characters and escape sequences are discarded.

Col keeps track of the character set as characters are read and makes sure the character set is correct when they are output.

If the input attempts to back up to the last flushed line, **col** will display a warning message.

SEE ALSO

`expand`(1), `nroff`(1), `tbl`(1)

STANDARDS

The **col** utility conforms to the Single UNIX Specification, Version 2. The **-l** option is an extension to the standard.

HISTORY

A **col** command appeared in Version 6 AT&T UNIX.

AVAILABILITY

The col command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/. COLCRT

NAME

colcrt — filter nroff output for CRT previewing

SYNOPSIS

colcrt [**-**] [**-2**] [*file ...*]

DESCRIPTION

Colcrt provides virtual half-line and reverse line feed sequences for terminals without such capability, and on which overstriking is destructive. Half-line characters and underlining (changed to dashing '–') are placed on new lines in between the normal output lines.

Available options:

-	Suppress all underlining. This option is especially useful for previewing *allboxed* tables from tbl(1).
-2	Causes all half-lines to be printed, effectively double spacing the output. Normally, a minimal space output format is used which will suppress empty lines. The program never suppresses two consecutive empty lines, however. The **-2** option is useful for sending output to the line printer when the output contains superscripts and subscripts which would otherwise be invisible.

EXAMPLES

A typical use of **colcrt** would be

```
tbl exum2.n | nroff -ms | colcrt - | more
```

SEE ALSO

nroff(1), troff(1), col(1), more(1), ul(1)

BUGS

Should fold underlines onto blanks even with the '**-**' option so that a true underline character would show.

Can't back up more than 102 lines.

General overstriking is lost; as a special case '|' overstruck with '–' or underline becomes '+'.

Lines are trimmed to 132 characters.

Some provision should be made for processing superscripts and subscripts in documents which are already double-spaced.

HISTORY

The **colcrt** command appeared in 3.0BSD.

AVAILABILITY

The colcrt command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/. COLRM

NAME

colrm — remove columns from a file

SYNOPSIS

colrm [*startcol* [*endcol*]]

DESCRIPTION

Colrm removes selected columns from a file. Input is taken from standard input. Output is sent to standard output.

If called with one parameter the columns of each line will be removed starting with the specified column. If called with two parameters the columns from the first column to the last column will be removed.

Column numbering starts with column 1.

SEE ALSO

awk(1), column(1), expand(1), paste(1)

HISTORY

The **colrm** command appeared in 3.0BSD.

AVAILABILITY

The colrm command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/.

DDATE

NAME

ddate – converts Gregorian dates to Discordian dates

SYNOPSIS

ddate [+format] [date]

DESCRIPTION

ddate prints the date in Discordian date format.

If called with no arguments, **ddate** will get the current system date, convert this to the Discordian date format and print this on the standard output. Alternatively, a Gregorian date may be specified on the command line, in the form of a numerical day, month and year.

If a format string is specified, the Discordian date will be printed in a format specified by the string. This mechanism works similarly to the format string mechanism of **date(1),** only almost completely differently. The fields are:

%A	Full name of the day of the week (i.e., Sweetmorn)
%a	Abbreviated name of the day of the week (i.e., SM)
%B	Full name of the season (i.e., Chaos)
%b	Abbreviated name of the season (i.e., Chs)
%d	Ordinal number of day in season (i.e., 23)
%e	Cardinal number of day in season (i.e., 23rd)
%H	Name of current Holyday, if any
%N	Magic code to prevent rest of format from being printed unless today is a Holyday.
%n	Newline
%t	Tab
%X	Number of days remaining until X-Day. (Not valid if the SubGenius options are not compiled in.)
%{	
%}	Used to enclose the part of the string which is to be replaced with the words "St. Tib's Day" if the current day is St. Tib's Day.
%.	Try it and see.

EXAMPLES

```
% ddate
Sweetmorn, Bureaucracy 42, 3161 YOLD
```

```
% ddate +'Today is %{%A, the %e of %B%}, %Y. %N%nCelebrate %H'
Today is Sweetmorn, the 42nd of Bureaucracy, 3161.
```

```
% ddate +"It's %{%A, the %e of %B%}, %Y. %N%nCelebrate %H" 26 9 1995
It's Prickle-Prickle, the 50th of Bureaucracy, 3161.
Celebrate Bureflux
```

```
% ddate +"Today's %{%A, the %e of %B%}, %Y. %N%nCelebrate %H" 29 2 1996
Today's St. Tib's Day, 3162.
```

BUGS

ddate(1) will produce undefined behaviour if asked to produce the date for St. Tib's day and its format string does not contain the St. Tib's Day delimiters %{ and %}.

NOTE

After 'X-Day' passed without incident, the Church of the SubGenius declared that it had got the year upside down - X-Day is actually in 8661 AD rather than 1998 AD. Thus, the True X-Day is Cfn 40, 9827.

AUTHOR

Original program by Druel the Chaotic aka Jeremy Johnson (mpython@gnu.ai.mit.edu)
Major rewrite by Lee H:. O:. Smith, KYTP, aka Andrew Bulhak (acb@dev.null.org)
Five tons of flax.

DISTRIBUTION POLICY

Public domain. All rites reversed.

SEE ALSO

date(1),
http://www.subgenius.com/
Malaclypse the Younger, *Principia Discordia, Or How I Found Goddess And What I Did To Her When I Found Her*

AVAILABILITY

The ddate command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/.

GETOPT

NAME

getopt – parse command options (enhanced)

SYNOPSIS

getopt optstring parameters **getopt** [options] [**−−**] optstring parameters **getopt** [options] **−o|−−options** optstring [options] [**−−**] parameters

DESCRIPTION

getopt is used to break up (*parse*) options in command lines for easy parsing by shell procedures, and to check for legal options. It uses the GNU **getopt**(3) routines to do this. The parameters **getopt** is called with can be divided into two parts: options which modify the way getopt will parse (*options* and *−o|−−options optstring* in the **SYNOPSIS),** and the parameters which are to be parsed (*parameters* in the **SYNOPSIS).** The second part will start at the first non−option parameter that is not an option argument, or after the first occurrence of '**−−**'. If no '**−o**' or '**−−options**' option is found in the first part, the first parameter of the second part is used as the short options string. If the environment variable **GETOPT_COMPATIBLE** is set, or if its first parameter is not an option (does not start with a '**−**', this is the first format in the **SYNOPSIS), getopt** will generate output that is compatible with that of other versions of **getopt**(1). It will still do parameter shuffling and recognize optional arguments (see section **COMPATIBILITY** for more information). Traditional implementations of **getopt**(1) are unable to cope with whitespace and other (shell−specific) special characters in arguments and non−option parameters. To solve this problem, this implementation can generate quoted output which must once again be interpreted by the shell (usually by using the **eval** command). This has the effect of preserving those characters, but you must call **getopt** in a way that is no longer compatible with other versions (the second or third format in the **SYNOPSIS).** To determine whether this enhanced version of **getopt**(1) is installed, a special test option (**−T**) can be used.

OPTIONS

−a, −−alternative
: Allow long options to start with a single '**−**'.

−h, −−help
: Output a small usage guide and exit successfully. No other output is generated.

−l, −−longoptions longopts
: The long (multi−character) options to be recognized. More than one option name may be specified at once, by separating the names with commas. This option may be given more than once, the *longopts* are cumulative. Each long option name in *longopts* may be followed by one colon to indicate it has a required argument, and by two colons to indicate it has an optional argument.

−n, −−name progname
: The name that will be used by the **getopt**(3) routines when it reports errors. Note that errors of **getopt**(1) are still reported as coming from getopt.

−o, −−options shortopts
: The short (one−character) options to be recognized. If this option is not found, the first parameter of **getopt** that does not start with a '**−**' (and is not an option argument) is used as the short options string. Each short option character in *shortopts* may be followed by one colon to indicate it has a required argument, and by two colons to indicate it has an optional argument. The first character of shortopts may be '**+**' or '**−**' to influence the way options are parsed and output is generated (see section **SCANNING MODES** for details).

−q, −−quiet
: Disable error reporting by getopt(3).

−Q, −−quiet−output
: Do not generate normal output. Errors are still reported by **getopt**(3), unless you also use *−q*.

−s, −−shell shell
: Set quoting conventions to those of shell. If no −s argument is found, the BASH conventions are used. Valid arguments are currently '**sh**' '**bash**', '**csh**', and '**tcsh**'.

−u, −−unquoted
: Do not quote the output. Note that whitespace and special (shell−dependent) characters can cause havoc in this mode (like they do with other **getopt**(1) implementations).

−T −−test
: Test if your **getopt**(1) is this enhanced version or an old version. This generates no output, and sets the error status to 4. Other implementations of **getopt**(1), and this version if the environment variable **GETOPT_COMPATIBLE** is set, will return '−−' and error status 0.

−V, −−version
: Output version information and exit successfully. No other output is generated.

PARSING

This section specifies the format of the second part of the parameters of **getopt** (the *parameters* in the **SYNOPSIS**). The next section (**OUTPUT**) describes the output that is generated. These parameters were typically the parameters a shell function was called with. Care must be taken that each parameter the shell function was called with corresponds to exactly one parameter in the parameter list of **getopt** (see the **EXAMPLES**). All parsing is done by the GNU **getopt**(3) routines. The parameters are parsed from left to right. Each parameter is classified as a short option, a long option, an argument to an option, or a non−option parameter. A simple short option is a '−' followed by a short option character. If the option has a required argument, it may be written directly after the option character or as the next parameter (ie. separated by whitespace on the command line). If the option has an optional argument, it must be written directly after the option character if present. It is possible to specify several short options after one '−', as long as all (except possibly the last) do not have required or optional arguments. A long option normally begins with '−−' followed by the long option name. If the option has a required argument, it may be written directly after the long option name, separated by '=', or as the next argument (ie. separated by whitespace on the command line). If the option has an optional argument, it must be written directly after the long option name, separated by '=', if present (if you add the '=' but nothing behind it, it is interpreted as if no argument was present; this is a slight bug, see the **BUGS**). Long options may be abbreviated, as long as the abbreviation is not ambiguous. Each parameter not starting with a '−', and not a required argument of a previous option, is a non−option parameter. Each parameter after a '−−' parameter is always interpreted as a non−option parameter. If the environment variable **POSIXLY_CORRECT** is set, or if the short option string started with a '+', all remaining parameters are interpreted as non−option parameters as soon as the first non−option parameter is found.

OUTPUT

Output is generated for each element described in the previous section. Output is done in the same order as the elements are specified in the input, except for non−option parameters. Output can be done in *compatible* (*unquoted*) mode, or in such way that whitespace and other special characters within arguments and non−option parameters are preserved (see **QUOTING**). When the output is processed in the shell script, it will seem to be composed of distinct elements that can be processed one by one (by using the shift command in most shell languages). This is imperfect in unquoted mode, as elements can be split at unexpected places if they contain whitespace or special characters. If there are problems parsing the parameters, for example because a required argument is not found or an option is not recognized, an error will be reported on stderr, there will be no output for the offending element, and a non−zero error status is returned. For a short option, a single '−' and the option character are generated as one parameter. If the option has an argument, the next parameter will be the argument. If the option takes an optional argument, but none was found, the next parameter will be generated but be empty in quoting mode, but no second parameter will be generated in unquoted (compatible) mode. Note that many other **getopt**(1) implementations do not support optional arguments. If several short options were specified after a single '−', each will be present in the output as a separate parameter. For a long option, '−−' and the full option name are generated as one parameter. This is done regardless whether the option was abbreviated or specified with a single '−' in the input. Arguments are handled as with short options. Normally, no non−option parameters output is generated until all options and their arguments have been generated. Then '−−' is generated as a single parameter, and after it the non−option parameters in the order they were found, each as a separate parameter. Only if the first character of the short options string was a '−', non−option parameter output is generated at the place they are found in the input (this is not supported if the first format of the **SYNOPSIS** is used; in that case all preceding occurrences of '−' and '+' are ignored).

QUOTING

In compatible mode, whitespace or 'special' characters in arguments or non−option parameters are not handled correctly. As the output is fed to the shell script, the script does not know how it is supposed to break the output into separate parameters. To circumvent this problem, this implementation offers quoting. The idea is that output is generated with quotes around each parameter. When this output is once again fed to the shell (usually by a shell **eval** command), it is split correctly into separate parameters. Quoting is not enabled if the environment variable **GETOPT_COMPATIBLE** is set, if the first form of the **SYNOPSIS** is used, or if the option '**−u**' is found. Different shells use different quoting conventions. You can use the '**−s**' option to select the shell you are using. The following shells are currently supported: '**sh**', '**bash**', '**csh**' and '**tcsh**'. Actually, only two 'flavors' are distinguished: sh−like quoting conventions and csh−like quoting conventions. Chances are that if you use another shell script language, one of these flavors can still be used.

SCANNING MODES

The first character of the short options string may be a '**−**' or a '**+**' to indicate a special scanning mode. If the first calling form in the **SYNOPSIS** is used they are ignored; the environment variable **POSIXLY_CORRECT** is still examined, though. If the first character is '**+**', or if the environment variable **POSIXLY_CORRECT** is set, parsing stops as soon as the first non−option parameter (ie. a parameter that does not start with a '**−**') is found that is not an option argument. The remaining parameters are all interpreted as non−option parameters. If the first character is a '**−**', non−option parameters are outputted at the place where they are found; in normal operation, they are all collected at the end of output after a '**−−**' parameter has been generated. Note that this '**−−**' parameter is still generated, but it will always be the last parameter in this mode.

COMPATIBILITY

This version of **getopt**(1) is written to be as compatible as possible to other versions. Usually you can just replace them with this version without any modifications, and with some advantages. If the first character of the first parameter of getopt is not a '**−**', getopt goes into compatibility mode. It will interpret its first parameter as the string of short options, and all other arguments will be parsed. It will still do parameter shuffling (ie. all non−option parameters are outputted at the end), unless the environment variable **POSIXLY_CORRECT** is set. The environment variable **GETOPT_COMPATIBLE** forces **getopt** into compatibility mode. Setting both this environment variable and **POSIXLY_CORRECT** offers 100% compatibility for 'difficult' programs. Usually, though, neither is needed. In compatibility mode, leading '**−**' and '**+**' characters in the short options string are ignored.

RETURN CODES

getopt returns error code **0** for successful parsing, **1** if **getopt**(3) returns errors, **2** if it does not understand its own parameters, **3** if an internal error occurs like out−of−memory, and **4** if it is called with **−T**.

EXAMPLES

Example scripts for (ba)sh and (t)csh are provided with the **getopt**(1) distribution, and are optionally installed in **/usr/share/getopt .**

ENVIRONMENT

POSIXLY_CORRECT

This environment variable is examined by the **getopt**(3) routines. If it is set, parsing stops as soon as a parameter is found that is not an option or an option argument. All remaining parameters are also interpreted as non−option parameters, regardless whether they start with a '**−**'.

GETOPT_COMPATIBLE

Forces **getopt** to use the first calling format as specified in the **SYNOPSIS**.

BUGS

getopt(3) can parse long options with optional arguments that are given an empty optional argument (but can not do this for short options). This **getopt**(1) treats optional arguments that are empty as if they were not present. The syntax if you do not want any short option variables at all is not very intuitive (you have to set them explicitly to the empty string).

AUTHOR

Frodo Looijaard <frodo@frodo.looijaard.name>

SEE ALSO

getopt(3), **bash**(1), **tcsh**(1).

AVAILABILITY

The getopt command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/. HEXDUMP

NAME

hexdump — ascii, decimal, hexadecimal, octal dump

SYNOPSIS

[**-bcCdovx**] [**-e** *format_string*] [**-f** *format_file*] [**-n** *length*] [**-s** *skip*] *file ...*

DESCRIPTION

The hexdump utility is a filter which displays the specified files, or the standard input, if no files are specified, in a user specified format.

The options are as follows:

-b *One-byte octal display*. Display the input offset in hexadecimal, followed by sixteen space-separated, three column, zero-filled, bytes of input data, in octal, per line.

-c *One-byte character display*. Display the input offset in hexadecimal, followed by sixteen space-separated, three column, space-filled, characters of input data per line.

-C *Canonical hex+ASCII display*. Display the input offset in hexadecimal, followed by sixteen space-separated, two column, hexadecimal bytes, followed by the same sixteen bytes in %_p format enclosed in "|" characters.

-d *Two-byte decimal display*. Display the input offset in hexadecimal, followed by eight space-separated, five column, zero-filled, two-byte units of input data, in unsigned decimal, per line.

-e *format_string*
Specify a format string to be used for displaying data.

-f *format_file*
Specify a file that contains one or more newline separated format strings. Empty lines and lines whose first non-blank character is a hash mark (**#**) are ignored.

-n *length*
Interpret only *length* bytes of input.

-o *Two-byte octal display*. Display the input offset in hexadecimal, followed by eight space-separated, six column, zero-filled, two byte quantities of input data, in octal, per line.

-s *offset*
Skip *offset* bytes from the beginning of the input. By default, *offset* is interpreted as a decimal number. With a leading **0x** or **0X**, *offset* is interpreted as a hexadecimal number, otherwise, with a leading **0**, *offset* is interpreted as an octal number. Appending the character **b**, **k**, or **m** to *offset* causes it to be interpreted as a multiple of 512, 1024, or 1048576, respectively.

-v The **-v** option causes hexdump to display all input data. Without the **-v** option, any number of groups of output lines, which would be identical to the immediately preceding group of output lines (except for the input offsets), are replaced with a line comprised of a single asterisk.

-x *Two-byte hexadecimal display*. Display the input offset in hexadecimal, followed by eight, space separated, four column, zero-filled, two-byte quantities of input data, in hexadecimal, per line.

For each input file, **hexdump** sequentially copies the input to standard output, transforming the data according to the format strings specified by the **-e** and **-f** options, in the order that they were specified.

Formats

A format string contains any number of format units, separated by whitespace. A format unit contains up to three items: an iteration count, a byte count, and a format.

The iteration count is an optional positive integer, which defaults to one. Each format is applied iteration count times.

The byte count is an optional positive integer. If specified it defines the number of bytes to be interpreted by each iteration of the format.

If an iteration count and/or a byte count is specified, a single slash must be placed after the iteration count and/or before the byte count to disambiguate them. Any whitespace before or after the slash is ignored.

The format is required and must be surrounded by double quote (" ") marks. It is interpreted as a fprintf-style format string (see `fprintf`(3)), with the following exceptions:

- An asterisk (*) may not be used as a field width or precision.
- A byte count or field precision *is* required for each "s" conversion character (unlike the `fprintf`(3) default which prints the entire string if the precision is unspecified).
- The conversion characters "h", "l", "n", "p" and "q" are not supported.
- The single character escape sequences described in the C standard are supported:

NUL	\0
<alert character>	\a
<backspace>	\b
<form-feed>	\f
<newline>	\n
<carriage return>	\r
<tab>	\t
<vertical tab>	\v

Hexdump also supports the following additional conversion strings:

_a[**dox**] Display the input offset, cumulative across input files, of the next byte to be displayed. The appended characters **d**, **o**, and **x** specify the display base as decimal, octal or hexadecimal respectively.

_A[**dox**] Identical to the **_a** conversion string except that it is only performed once, when all of the input data has been processed.

_c Output characters in the default character set. Nonprinting characters are displayed in three character, zero-padded octal, except for those representable by standard escape notation (see above), which are displayed as two character strings.

_p Output characters in the default character set. Nonprinting characters are displayed as a single "`.`".

_u Output US ASCII characters, with the exception that control characters are displayed using the following, lower-case, names. Characters greater than 0xff, hexadecimal, are displayed as hexadecimal strings.

000 nul	001 soh	002 stx	003 etx	004 eot	005 enq
006 ack	007 bel	008 bs	009 ht	00A lf	00B vt
00C ff	00D cr	00E so	00F si	010 dle	011 dc1
012 dc2	013 dc3	014 dc4	015 nak	016 syn	017 etb
018 can	019 em	01A sub	01B esc	01C fs	01D gs
01E rs	01F us	0FF del			

The default and supported byte counts for the conversion characters are as follows:

`%_c, %_p, %_u, %c`	One byte counts only.
`%d, %i, %o, %u, %X, %x`	Four byte default, one, two and four byte counts supported.

`%E, %e, %f, %G, %g`	Eight byte default, four byte counts supported.

The amount of data interpreted by each format string is the sum of the data required by each format unit, which is the iteration count times the byte count, or the iteration count times the number of bytes required by the format if the byte count is not specified.

The input is manipulated in ''blocks'', where a block is defined as the largest amount of data specified by any format string. Format strings interpreting less than an input block's worth of data, whose last format unit both interprets some number of bytes and does not have a specified iteration count, have the iteration count incremented until the entire input block has been processed or there is not enough data remaining in the block to satisfy the format string.

If, either as a result of user specification or hexdump modifying the iteration count as described above, an iteration count is greater than one, no trailing whitespace characters are output during the last iteration.

It is an error to specify a byte count as well as multiple conversion characters or strings unless all but one of the conversion characters or strings is **_a** or **_A**.

If, as a result of the specification of the **−n** option or end-of-file being reached, input data only partially satisfies a format string, the input block is zero-padded sufficiently to display all available data (i.e. any format units overlapping the end of data will display some number of the zero bytes).

Further output by such format strings is replaced by an equivalent number of spaces. An equivalent number of spaces is defined as the number of spaces output by an **s** conversion character with the same field width and precision as the original conversion character or conversion string but with any "+", " ", "#" conversion flag characters removed, and referencing a NULL string.

If no format strings are specified, the default display is equivalent to specifying the **−x** option.

hexdump exits 0 on success and >0 if an error occurred.

EXAMPLES

Display the input in perusal format:

```
"%06.6_ao "  12/1 "%3_u "
"\t\t" "%_p "
"\n"
```

Implement the −x option:

```
"%07.7_Ax\n"
"%07.7_ax  " 8/2 "%04x " "\n"
```

STANDARDS

The **hexdump** utility is expected to be IEEE Std 1003.2 ("POSIX.2") compatible.

AVAILABILITY

The hexdump command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/.

LINE

NAME

line – read one line

SYNOPSIS

line

DESCRIPTION

The utility *line* copies one line (up to a newline) from standard input to standard output. It always prints at least a newline and returns an exit status of 1 on EOF or read error.

SEE ALSO

read(1)

AVAILABILITY

The line command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/.

RENAME

NAME

rename − Rename files

SYNOPSIS

rename *from to file...*
rename *-V*

DESCRIPTION

rename will rename the specified files by replacing the first occurrence of *from* in their name by *to*.

−V, −−version
Display version information and exit.

For example, given the files
foo1, ..., *foo9*, *foo10*, ..., *foo278*, the commands

```
rename foo foo0 foo?
rename foo foo0 foo??
```

will turn them into *foo001*, ..., *foo009*, *foo010*, ..., *foo278*. And

```
rename .htm .html *.htm
```

will fix the extension of your html files.

SEE ALSO

mmv(1), **mv**(1)

AVAILABILITY

The rename command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/. REV

NAME

rev — reverse lines of a file or files

SYNOPSIS

rev [*file ...*]

DESCRIPTION

The **rev** utility copies the specified files to the standard output, reversing the order of characters in every line. If no files are specified, the standard input is read.

AVAILABILITY

The rev command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/.

TAILF

NAME

tailf – follow the growth of a log file

SYNOPSIS

tailf [*OPTION*] *file*

DESCRIPTION

tailf will print out the last 10 lines of a file and then wait for the file to grow. It is similar to **tail -f** but does not access the file when it is not growing. This has the side effect of not updating the access time for the file, so a filesystem flush does not occur periodically when no log activity is happening.

tailf is extremely useful for monitoring log files on a laptop when logging is infrequent and the user desires that the hard disk spin down to conserve battery life.

Mandatory arguments to long options are mandatory for short options too.

−n, **−−lines**=*N*, **−N**
output the last *N* lines, instead of the last 10.

AUTHOR

This program was originally written by Rik Faith (faith@acm.org) and may be freely distributed under the terms of the X11/MIT License. There is ABSOLUTELY NO WARRANTY for this program. The latest inotify based implementation was written by Karel Zak (kzak@redhat.com).

SEE ALSO

tail(1), **less**(1)

AVAILABILITY

The tailf command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/.

WHEREIS

NAME

whereis – locate the binary, source, and manual page files for a command

SYNOPSIS

whereis [**–bmsu**] [**–BMS** *directory*. . . **–f**] *filename* . . .

DESCRIPTION

whereis locates source/binary and manuals sections for specified files. The supplied names are first stripped of leading pathname components and any (single) trailing extension of the form ***.ext,*** for example, **.c**. Prefixes of **s.** resulting from use of source code control are also dealt with. **whereis** then attempts to locate the desired program in a list of standard Linux places.

OPTIONS

–b Search only for binaries.

–m Search only for manual sections.

–s Search only for sources.

–u Search for unusual entries. A file is said to be unusual if it does not have one entry of each requested type. Thus '**whereis –m –u** *' asks for those files in the current directory which have no documentation.

–B Change or otherwise limit the places where **whereis** searches for binaries.

–M Change or otherwise limit the places where **whereis** searches for manual sections.

–S Change or otherwise limit the places where **whereis** searches for sources.

–f Terminate the last directory list and signals the start of file names, and *must* be used when any of the **–B**, **–M**, or **–S** options are used.

EXAMPLE

Find all files in **/usr/bin** which are not documented in **/usr/man/man1** with source in **/usr/src**:

```
example% cd /usr/bin
example% whereis –u –M /usr/man/man1 –S /usr/src –f *
```

FILES

/{bin,sbin,etc}

/usr/{lib,bin,old,new,local,games,include,etc,src,man,sbin,
X386,TeX,g++-include}

/usr/local/{X386,TeX,X11,include,lib,man,etc,bin,games,emacs}

SEE ALSO

chdir(2V)

BUGS

Since **whereis** uses **chdir**(2V) to run faster, pathnames given with the **–M**, **–S**, or **–B** must be full; that is, they must begin with a '/'.

whereis has a hard-coded path, so may not always find what you're looking for.

AVAILABILITY

The whereis command is part of the util-linux-ng package and is available from ftp://ftp.kernel.org/pub/linux/utils/util-linux-ng/.