sort

Sort text files.
Sort, merge, or compare all the lines from the files given (or standard input.)

```
Syntax
      sort [options] [file...]
      sort --help
      sort --version

Options
   sort has three modes of operation:
     Sort (the default), Merge (-m), and Check(-c)

       -c     Check whether the given files are already sorted:  if  they  are
              not all sorted, print an error message and exit with a status of 1.

       -m     Merge the given files by sorting them as a group. Each input
              file should already be individually sorted.  It always works to
              sort instead of merge; merging is provided because it is faster,
              in the case where it works.
```

The following options affect the **ordering** of output lines. They may be specified globally or as part of a specific key field. If no key fields are specified, global options apply to comparison of entire lines; otherwise the global options are inherited by key fields that do not specify any special options of their own. The `-b', `-d', `-f' and `-i' options classify characters according to the `LC_CTYPE' locale.

```
`-b'
     Ignore leading blanks when finding sort keys in each line.

`-d'
     Sort in "phone directory" order: ignore all characters except
     letters, digits and blanks when sorting.

`-f'
     Fold lowercase characters into the equivalent uppercase characters
     when sorting so that, for example, `b' and `B' sort as equal.

`-g'
     Sort numerically, using the standard C function `strtod' to convert
     a prefix of each line to a double-precision floating point number.
     This allows floating point numbers to be specified in scientific
     notation, like `1.0e-34' and `10e100'.  Do not report overflow,
     underflow, or conversion errors.  Use the following collating
     sequence:

        * Lines that do not start with numbers (all considered to be
          equal).

        * NaNs ("Not a Number" values, in IEEE floating point
          arithmetic) in a consistent but machine-dependent order.

        * Minus infinity.

        * Finite numbers in ascending numeric order (with -0 and +0
          equal).

        * Plus infinity.

     Use this option only if there is no alternative; it is much slower
     than `-n' and it can lose information when converting to floating
     point.

`-i'
     Ignore unprintable characters.

`-M'
     An initial string, consisting of any amount of whitespace, followed
```

by a month name abbreviation, is folded to UPPER case and compared in the order `JAN' < `FEB' < ... < `DEC'. Invalid names compare low to valid names. The `LC_TIME' locale determines the month spellings.

`-n'
Sort numerically: the number begins each line; specifically, it consists of optional whitespace, an optional `-' sign, and zero or more digits possibly separated by thousands separators, optionally followed by a radix character and zero or more digits. The `LC_NUMERIC' locale specifies the radix character and thousands separator.

`sort -n' uses what might be considered an unconventional method to compare strings representing floating point numbers. Rather than first converting each string to the C `double' type and then comparing those values, sort aligns the radix characters in the two strings and compares the strings a character at a time. One benefit of using this approach is its speed. In practice this is much more efficient than performing the two corresponding string-to-double (or even string-to-integer) conversions and then comparing doubles. In addition, there is no corresponding loss of precision. Converting each string to `double' before comparison would limit precision to about 16 digits on most systems.

Neither a leading `+' nor exponential notation is recognized. To compare such strings numerically, use the `-g' option.

`-r'
Reverse the result of comparison, so that lines with greater key values appear earlier in the output instead of later.

Other options:

`-o *OUTPUT-FILE*'
Write output to *OUTPUT-FILE* instead of standard output. If *OUTPUT-FILE* is one of the input files, `sort' copies it to a temporary file before sorting and writing the output to *OUTPUT-FILE*.

`-t *SEPARATOR*'
Use character *SEPARATOR* as the field separator when finding the sort keys in each line. By default, fields are separated by the empty string between a non-whitespace character and a whitespace character. That is, given the input line ` foo bar', `sort' breaks it into fields ` foo' and ` bar'. The field separator is not considered to be part of either the field preceding or the field following.

`-u'
For the default case or the `-m' option, only output the first of a sequence of lines that compare equal. For the `-c' option, check that no pair of consecutive lines compares equal.

`-k *POS1*[,*POS2*]'
The recommended, POSIX, option for specifying a sort field. The field consists of the part of the line between POS1 and POS2 (or the end of the line, if POS2 is omitted), _inclusive_. Fields and character positions are numbered starting with 1. So to sort on the second field, you'd use `-k 2,2' See below for more examples.

`-z'
Treat the input as a set of lines, each terminated by a zero byte (ASCII NUL (Null) character) instead of an ASCII LF (Line Feed). This option can be useful in conjunction with `perl -0' or `find -print0' and `xargs -0' which do the same in order to reliably handle arbitrary pathnames (even those which contain Line Feed characters.)

`+*POS1*[-*POS2*]'

```
      The obsolete, traditional option for specifying a sort field.
      The field consists of the line between POS1 and up to but _not
      including_ POS2 (or the end of the line if POS2 is omitted).
      Fields and character positions are numbered starting with 0.
      See below.

`--help'

`--version'
```

HOW LINES ARE COMPARED

A pair of lines is compared as follows: if any key fields have been specified, `sort' compares each pair of fields, in the order specified on the command line, according to the associated ordering options, until a difference is found or no fields are left. Unless otherwise specified, all comparisons use the character collating sequence specified by the `LC_COLLATE' locale.

If any of the global options `Mbdfinr' are given but no key fields are specified, `sort' compares the entire lines according to the global options.

Finally, as a last resort when all keys compare equal (or if no ordering options were specified at all), `sort' compares the entire lines. The last resort comparison honors the `-r' global option. The `-s' (stable) option disables this last-resort comparison so that lines in which all fields compare equal are left in their original relative order. If no fields or global options are specified, `-s' has no effect.

GNU `sort' (as specified for all GNU utilities) has no limits on input line length or restrictions on bytes allowed within lines. In addition, if the final byte of an input file is not a newline, GNU `sort' silently supplies one. A line's trailing newline is part of the line for comparison purposes; for example, with no options in an ASCII locale, a line starting with a tab sorts before an empty line because tab precedes newline in the ASCII collating sequence.

Upon any error, `sort' exits with a status of `2'.

If the environment variable `TMPDIR' is set, `sort' uses its value as the directory for temporary files instead of `/tmp'. The `-T TEMPDIR' option in turn overrides the environment variable.

NOTES

Historical (BSD and System V) implementations of `sort' have differed in their interpretation of some options, particularly `-b', `-f', and `-n'. GNU sort follows the POSIX behavior, which is usually (but not always!) like the System V behavior. According to POSIX, `-n' no longer implies `-b'. For consistency, `-M' has been changed in the same way. This may affect the meaning of character positions in field specifications in obscure cases. The only fix is to add an explicit `-b'.

A position in a sort field specified with the `-k' or `+' option has the form `F.C', where F is the number of the field to use and C is the number of the first character from the beginning of the field (for `+POS') or from the end of the previous field (for `-POS'). If the `.C' is omitted, it is taken to be the first character in the field. If the `-b' option was specified, the `.C' part of a field specification is counted from the first nonblank character of the field (for `+POS') or from the first nonblank character following the previous field (for `-POS').

A sort key option may also have any of the option letters `Mbdfinr' appended to it, in which case the global ordering options are not used for that particular field. The `-b' option may be independently attached to either or both of the `+POS' and `-POS' parts of a field specification, and if it is inherited from the global options it will be attached to both. Keys may span multiple fields.

Examples

```
Character Sort:

$ sort countries.txt

Numeric sort:

$ sort -n numbers.txt

To sort the file below on the third field (area code):
```

Jim Alchin 212121 Seattle
Bill Gates 404404 Seattle
Steve Jobs 246810 Nevada
Scott Neally 212277 Los Angeles

```
$ sort -k 3,3 people.txt> sorted.txt

or using the 'old' syntax:
$ sort +2 -3 people.txt> sorted2.txt
```

```
To sort the same file on the 4th column and supress duplicates: (should return 3 rows)
$ sort -u -k 4,4 people.txt> sorted3.txt
```

In the remaining examples, the POSIX `-k' option is used to specify sort keys rather than the obsolete `+POS1-POS2' syntax.

```
Sort in descending (reverse) numeric order:

$ sort -nr

Sort alphabetically, omitting the first and second fields.  This
uses a single key composed of the characters beginning at the
start of field three and extending to the end of each line:

$ sort -k3

Sort numerically on the second field and resolve ties by sorting
alphabetically on the third and fourth characters of field five.
Use `:' as the field delimiter:

$ sort -t : -k 2,2n -k 5.3,5.4
```

Note that if you had written `-k 2' instead of `-k 2,2' `sort' would have used all characters beginning in the second field and extending to the end of the line as the primary _numeric_ key. For the large majority of applications, treating keys spanning more than one field as numeric will not do what you expect.

Also note that the `n' modifier was applied to the field-end specifier for the first key. It would have been equivalent to specify `-k 2n,2' or `-k 2n,2n'. All modifiers except `b' apply to the associated _field_, regardless of whether the modifier character is attached to the field-start and/or the field-end part
of the key specifier.

```
Sort the password file on the fifth field and ignore any leading
white space.  Sort lines with equal values in field five on the
numeric user ID in field three:

$ sort -t : -k 5b,5 -k 3,3n /etc/passwd

An alternative is to use the global numeric modifier `-n':

$ sort -t : -n -k 5b,5 -k 3,3 /etc/passwd

Generate a tags file in case insensitive sorted order:

$ find src -type f -print0 | sort -t / -z -f | xargs -0 etags --append
```

The use of `-print0', `-z', and `-0' in this case mean that pathnames that contain Line Feed characters will not get broken up by the sort operation.

Finally, to ignore both leading and trailing white space, you could have applied the `b' modifier to the field-end specifier for the first key,

```
$ sort -t : -n -k 5b,5b -k 3,3 /etc/passwd

or by using the global `-b' modifier instead of `-n' and an
explicit `n' with the second key specifier:

$ sort -t : -b -k 5,5 -k 3,3n /etc/passwd
```

A file name of `-' means standard input.

By default, sort writes the results to the standard output.

"We never sit anything out. We are cups, constantly and quietly being filled. The trick is, knowing how to tip ourselves over and let the Beautiful Stuff out" ~ Ray Bradbury

Related:

head - Output the first part of file(s)
nl - Number lines and write files
printf - Format and print data
Equivalent Windows commands: SORT - Sort input