What's GNU, Part Five: sort

By Jerry Peek

This month, in the fifth article of a series on new features added to utilities by GNU programmers and others, let's look at *sort*. What's new? There are new ways to sort that aren't just lexicographic. For example, you can sort on month abbreviations ("Jan," "Jun") and general numeric formats. You can also handle NUL-terminated records — great for sorting filenames to pass to *xargs*. (All examples are based on GNU *sort version 5.2.1* from the *Debian stable* distribution.)

Forget What You Know

If you learned to use *sort* quite a while ago, you may not recognize its most recent incarnations. Some command-line options have changed, and the GNU version no longer truncates long lines of data — a longstanding bug in early versions of the utility. Additionally, GNU *sort* now follows *POSIX* rules, which are mostly the same as the older *System V* rules, but fairly different than the old *BSD* rules.

Now, for instance, your `LC_COLLATE` locale setting affects sorting order, while older versions of *sort* assumed native byte order.

To get the old sort order, set the `LC_ALL` environment variable to `C`. *Listing One* shows an example: the first command shows an unsorted file; the second command temporarily sets `LC_ALL` to `C` and sorts the file in the old native byte order. The third command sorts the file with the system default, here `en_US`, `ISO-8859-1`.

Playing the "Field"

By default, *sort* reorders data by comparing entire lines, where a "line" is the text between two newline (the ASCII Linefeed) characters. The newlines don't participate in the comparison. However, *sort* has always been able to sort lines using one or more of the *fields* in each line as sort keys.

By default (unless you use the `-t` or `--field-separator` option), the field separator is the empty string between a non-blank character and a blank character. (The keyword `blank` in the `LC_CTYPE` locale lists the characters that are "blank." The default characters are Space and Tab.) This imaginary point isn't actually part of the data, but thinking of it as real can help clarify what's going on.

For example, it's possible for a field to start with a number of blanks — especially in a file that uses Space or Tab characters to make columns. To demonstrate, the first field of the first line of sample data below starts with two blanks and the third field starts with eight:

```
  99         Smith  N
 100      Appleton  Y
```

Whether those blanks affect the sort depends on what the other (non-blank) data is and on the collation order in your locale. (This new reality, that spaces won't necessarily sort before letters, can be hard for some long-time users and a certain *Linux Magazine* columnist to remember.) As always, the `-b` option tells *sort* to ignore leading blanks and not to include them in field comparisons.

In the old version of *sort*, the numeric-sort option `-n` would automatically set `-b`. The POSIX version doesn't do this, though; you have to specify `-b` explicitly.

Choosing the Sort Key(s)

Command-line options like `+`*m*`-`*n* told old *sort* not to sort on the entire line, but to start a sort key at field *m* and end it just before field *n*. The first field on a line was numbered 0. So, `sort +0 -1` told *sort* to sort on only the first field — that is, the sort key starts at field zero and ends just before field one.

Newer versions use the options `-k` *m*`[,`*n*`]` or `--key=`*m*`[,`*n*`]` instead. In this new scheme, the key is the part of the line between the start of the field *m* and the end of the field *n*. The first field is numbered 1. So, `-k 1,1` sorts on only the first field — that is, starting at field 1 and ending at field 1.

On both the old and new systems, omitting the ending field tells *sort* to compare all fields from the first field specified through the last field on the line (not including the

LISTING ONE: The effects of changing locale

```
$ cat sortme1
Baby
Apple
baby
apple
$ LC_ALL=C sort sortme1
Apple
Baby
apple
baby
$ sort sortme1
apple
Apple
baby
Baby
```

LISTING TWO: Example of general numeric sort

```
$ cat sortme2
1.65e-12
-1.65e-12
+1.65e-12
0xFF
23
9E1
9.99e99
9.99e-99
$ sort -g  sortme2
-1.65e-12
9.99e-99
+1.65e-12
1.65e-12
23
9E1
0xFF
9.99e99
```

newline character). So, the old `sort +2` or the new `sort -k 3` sorts on all fields from the third to the last.

In both the old and the new versions, you can specify which character positions in a field are part of the sort key by adding a period and a number to the end of a field specification. But again, the old and new systems have differences. Here are the old and new ways, respectively, to sort *file* with the sort key as the second non-blank character of the second field:

```
$ sort +1.1b -1 .2b file
$ sort -k  2.2b,2.2b file
```

As always, you can add flags like `-b` or `-f` (but without the hyphen) after a field specification to apply the flag just to the specified fields. (See the example in *Listing Three*.)

Playing the Numbers

Plain *sort* sorts fields containing numbers, but not always well. The sort isn't exactly "numeric," because, by default, *sort* doesn't know the difference between digit characters and other characters. For instance, `62` can sort before `6.2` and `6` before `-6`.

Old *sort* had the `-n` option to control this, but it assumed that everyone used a dot (`.`) between the whole and fractional parts of a number. (A European who wrote six and a half as `6,5` would have been out of luck.) Now, *sort* finds the thousands separator and the decimal-point characters from the `LC_NUMERIC` locale.

The GNU version, at least, uses a clever trick to speed sorting and also to avoid rounding errors on floating-point numbers. The info page describes it:

> *Rather than first converting each string to the C 'double' type and then comparing those values,* sort *aligns the decimal-point 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 a double before comparison would limit precision to about 16 digits on most systems.*

As was mentioned already, the `-b` option to ignore leading blanks isn't automatic anymore when you use `-n`. You have to specify it explicitly.

Numbers in hex or scientific notation, like `0x1234` or `6.5e-10`, were hard or impossible to sort with older versions of *sort*. Nowadays, though, your version probably has the `-g` option to do a general numeric sort. This converts numbers to a double float representation using `strtod()`. It understands upper and lowercase *inf* or *infinity*, as well as *nan* or *NAN* ("not a number") followed by an optional string in parentheses.

LISTING THREE: Sorting by date

```
1$ cat sortme3
Jun  3 2005 Rehired
Jun 13 2005 Fired
Mar 15 2005 Fired
Mar 16 2003 Hired
Mar  6 2004 Promoted
2$ sort -k  3,3 sortme3
Mar 16 2003 Hired
Mar  6 2004 Promoted
Jun 13 2005 Fired
Jun  3 2005 Rehired
Mar 15 2005 Fired
3$ sort -k  3,3 -k  1,1M sortme3
Mar 16 2003 Hired
Mar  6 2004 Promoted
Mar 15 2005 Fired
Jun 13 2005 Fired
Jun  3 2005 Rehired
4$ sort -k  3,3 -k  1,1M -k  2,2b sortme3
Mar 16 2003 Hired
Mar  6 2004 Promoted
Mar 15 2005 Fired
Jun 13 2005 Fired
Jun  3 2005 Rehired
5$ sort -k  3,3 -k  1,1M -k  2,2nb sortme3
Mar 16 2003 Hired
Mar  6 2004 Promoted
Mar 15 2005 Fired
Jun  3 2005 Rehired
Jun 13 2005 Fired
```

Unfortunately, unlike `strtod()`, *sort* doesn't report overflow, underflow, or conversion errors. Also remember that conversion to a double means that small differences can be lost due to rounding errors.

Output sorted by `-g` begins with lines that don't start with numbers, then nan's, minus infinity, finite numbers, and plus infinity. *Listing Two* shows a general numeric sort.

Sorting Dates

The `-M` flag sorts by month name abbreviation. `Jan` (for January) comes before `Feb`, and so on. This works for non-English languages if the `LC_TIME` locale category is set properly. Upper- and lowercase compare equally, and leading blanks are ignored.

Listing Three has an extended example that also demonstrates `-k`. The shell prompts are numbered.

Command 1 shows the file; it holds a jumbled personnel record. Command 2 sorts the third field, the year. The records for 2005 are wrong, so command 3 includes a second sort key, the month (field 1). Putting `M` at the end of the field specification means the month sort happens only in this field. Next, command 4 tries to sort the day of the month, specifying the `b` flag so the initial spaces are ignored (some days have more leading spaces than others).

That doesn't do the job because `13` sorts before `3` lexicographically. To specify a numeric sort for the day, use `nb` instead of just `b`.

Here's another way to describe the effect of multiple sort keys. First, sort by year (field 3). A tie in the year (on any records that have the same year) is broken by sorting on the month (field 1). A tie in the year and month is broken by sorting on the date (field 2).

NUL field and record separators

Last but not least is a very handy technique for sorting arbitrary data. GNU *sort* can handle NUL (all-zero) bytes. NUL bytes aren't common in textual data and they're illegal in Linux filenames. That makes NUL bytes a reliable way to delimit fields and records. (Several other GNU utilities can output NULs between fields or records.)

For instance, if you have a NUL-separated list of pathnames from `sort -print0`, using `sort -z` tells *sort* to use a NUL record (line) separator. To specify NUL field separators, use `sort --field-separator='\0'` or `sort -t '\0'`.

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for 25 years. He's happy to hear from readers; see http://www.jpeek.com/contact.html.