by PETER COLLINSON, Hillside Systems

# Sorting

Every so often, I try to write articles that contain practical help. Actually, it seems these are the articles that generate the most email. I should reiterate that mail from readers is very welcome; it often sparks new articles or follow-ups. I try hard to place correct information in these columns, but obviously sometimes I do miss the mark. So, if you write correcting something that I expressed badly or have simply got wrong, I will suggest to the editor that your mail is printed in Letters to the Editor. It is worth sending mail that fixes bugs in the articles; it means other readers get corrected data and also educates me.

If I ask for your mail to be published, I will always give you a chance to fix the prose in your often hastily written note, so it's quite possible to dash off some corrective email without feeling that I will instantly pass it on to be printed. I cannot promise that the editor will have space to print your letter, but I don't think that there have been many cases where a letter that I suggested for inclusion has been "spiked," as the newspaper people say.

The notion to write this column came when I was processing data from one of the logs created by my Web server. The referrer log tells me the last port of call made by a visiting surfer; it records the page that a visitor was reading when they clicked a hyperlink to access one of my pages or images. I can find out who has links to my pages, and how often those links are exercised. The log has one line per record and three fields per line: the source URL, an arrow (in the form of `->`) and the destination page or image. Each field is separated by white space and so can easily be processed with standard UNIX tools.

There are several questions that I like to ask about this data. What pages are most frequently used to link to my pages? Which pages are most frequently accessed from outside? Are there any surprise entry points that people have linked to? Is anyone linking directly to an image? I do think that it's bad manners to incorporate other people's images in your pages, unless you inform and ask the originator. On the whole, I don't mind people using my images as long as proper credit is given and others are not passing off my work as their own.

## Using awk

I haven't created any specific shell scripts to process the referrer log data. I usually type the lines that I need into the shell to obtain the result I am looking for. The questions above mean that I will be processing one or other of the space-separated data columns from the file, and I need some way of selecting the appropriate data. The tool that I always use to select one column from a file is `awk`. For example, the command

```
$ awk '{print $3}' log
```

will print column three of the source data file log on the standard output of the command. This column holds the target page addresses in the referrer log. The `awk` program in braces is quoted with single quotes so that the dollar character is not

MICHELLE FRIESENHAHN WILBY

expanded by the shell. Using quotes is a good habit to get into for awk or sed programs typed on the command line.

I find that I use the column selection facilities of awk more frequently than any of its other commands. The column selection format is easy to remember and is useful in many circumstances. For example,

```
$ awk '{print $3,$1}' log
```

can be used to filter the arrow out of the referrer log and also swaps columns.

The general form of an awk command is

*selector { commands }*

The selector is applied to all the lines that are read from the input data file, and if the selection succeeds, then the commands within the braces are executed. In the above examples, the selector is empty so the single command is applied to all the lines from the source file log.

Actually, because the awk command is processing a log file, it's a good idea to be suspicious of the data. Sometimes log files are not written properly, or the data that is written onto them is not what you expect. In this case, being suspicious paid off. It turns out that some of the URLs written in column one of the log contain embedded spaces, meaning that rather than having three columns on the file, there are sometimes more. I used a small one-liner to determine this:

```
$ awk 'NF != 3 {print $0}' log
```

The command makes use of the selector field. The selector here is a Boolean expression testing the inbuilt awk variable NF that holds the number of fields that awk has found in the input line. If the number of fields is not equal to three, then I print the whole line ($0). I then look at the output to see what the exceptions are and quickly see the embedded spaces in the five or so lines that are output.

However, because the extra spaces are only in column one of the log, I can use an alternative procedure to obtain the last column. The easiest way to pick off the last column in a file is to use

```
$ awk '{print $NF}' log
```

Because the NF variable is set to the number of fields on the input line, $NF will pick off the last column on each line. However, things are a little trickier if I want to accurately pick off the first column in the file. The command

```
$ awk '{print $1}' log
```

won't work properly because the data that should be in column one is sometimes spread into two or more columns. It's possible to make awk do the right thing by sticking the columns together using a sequence of commands that feed data into the awk command.

However, life is too short and the answers to the questions that I posed above are not materially affected by the slight inaccuracies caused by occasional failure to pick up the complete URL from each line. So my judgment in this case is that using the `$1` selector to pick off the first column is good enough.

## The uniq Command

Well, having selected the data, what next? I can look at the raw data, but that's tedious. It's better to process it somehow to reduce the amount of information that I need to look through. One obvious way of looking at data with replicated entries is to generate output that omits duplicates. The `uniq` command can do this.

The `uniq` command is designed to output exactly one instance of any lines that are repeated in its input stream, so simply using the raw command as a filter will ensure that we only see each data item once. However, I prefer to see output that contains the data and the number of times that a particular data item is repeated. When `uniq` is supplied with the `-c` switch, it will print a count of each repeated line along with the data. So if we have a data stream like

```
one
two
two
three
three
three
```

then `uniq -c` will print

```
   1 one
   2 two
   3 three
```

counting the frequency that each repeated line occurs. However, to use this counting ability on raw data, we first need to sort the data so that all identical lines are next to each other. We'll type

```
$ awk '{print $NF}' | sort | uniq -c
```

and will see the frequency counts and the data being output. The output from this command line is still not as helpful as it might be, so as a final touch, I will usually pass the data into the `sort` command again. This time I will tell the command to sort numerically on the first field, the count, by supplying the `-n` switch. So, when typing the line above, I'll usually append

```
... | sort -n
```

or

```
... | sort -nr
```

depending on how I want the output to be sorted. The first variant of the command will sort into ascending numerical order, so the items with the most access frequency will be printed last. I usually prefer the data to be sorted into descending frequency order and use the `r` option to the `sort` command to reverse the order of the sort. It's often true to say that the data of most interest is at the start and the end of the frequency spectrum.

So processing column three of the referrer log to generate a frequency list will answer some of my original questions, the queries about the targets of other people's links. I can also use `grep` to find out about direct links to images from this output, and identify the source of the direct links to images by looking through the original data.

*If the visitor came to my pages from a search engine, then the arguments to the search are contained at the end of the URL, after a question mark. This can be helpful. I can deduce what people were looking for in the search engines when they ended up at my pages.*

To answer the other questions about the source of links, I'll need to look at column one of the log, the source URL. When looking at column one of the data, it helps to know a little about URL construction. If the visitor came to my pages from a search engine, then the arguments to the search are contained at the end of the URL, after a question mark. This can be helpful. I can deduce what people were looking for in the search engines when they ended up at my pages.

However, if I am examining the first column to deduce the most frequently used links to my pages, I am not interested in the search information, I just want to see the base URL. The trick is to massage the data from the log file before it's passed into `sort`, `uniq` and `sort -n`. Simply deleting any data after a question mark will ensure that only the URL is left from the source data, so I'll type something like

```
$ awk '{print $1}' log | sed -e 's/?.*$//' |
      sort | uniq -c | sort -nr
```

Here, the argument to `sed` deletes all data on any input line that starts at a question mark and extends to the end of the line. Using this prepass technique to suppress or alter the data, there are various other refinements that I can make to give me the answers to other questions that I may ask.

Using `uniq -c` and `sort` to obtain frequency counts of data is applicable in many situations. It's a common method to reduce information to a form that allows you to appreci-

ate the raw data in a more meaningful way. For example, the `du` command generates two columns of data: a number relating to the disk occupancy of the directory and the name of the directory. You can sort this output using `sort -n` to give yourself a feel for what is happening on the file system. However, beware that the `du` command prints the sizes in kilobytes on SunOS and in 512-byte units on Solaris. The trick here is to use the `-k` switch to `du` on Solaris.

### More on sort

If you look at the manual page on your system, you'll find that the `sort` command has several useful switches that can help you look at data in different ways. For example, the `-f` option treats lower- and uppercase letters the same for sorting, and can be helpful if you want to look at a directory using `ls`, placing files of similar names next to each other.

```
$ ls | sort -f
```

This command will revert to the original form of `ls`, giving you one file name per line. If you want a columnar listing, then pass the output from the above command through `pr`, supplying the `-t` option to suppress headers:

```
... | pr -t -6
```

The `-6` will give you six columns. My shell has an environment variable called `COLUMNS` that is set to the number of columns in the current window, so

```
... | pr -t -8 -w $COLUMNS
```

will print eight columns across the width of the screen. You do need to be careful of the output from `pr`; it will truncate data that doesn't fit into its precomputed column space, so file names can appear shorter than they are.

Of course, it's harder to sort the output of an `ls -l` command into case-independent name order, because the output lines have all the other file information, such as file permissions, owner identity, size and so on. By default, if you just sling the data from the `ls -l` command into the `sort` program, then the whole line will be used as a key and the output will probably not be what you want. You need to tell the `sort` command to use some part of the input line as the sorting key.

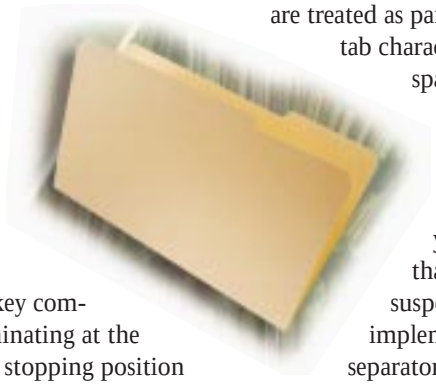
How the user specifies the sorting key is an area that has changed in Solaris from SunOS because, I suspect, the POSIX standard defines some new syntax. The POSIX committee meddled with existing practice, largely because the original method of specifying sort keys was hard to grasp and was prone to error. In fact, there was already some divergence in the System V world from the original specification that was in use in the BSD (and hence the SunOS) world. Also, the original syntax assigned special meaning to arguments starting with `+` and `-`, and the POSIX committee has tried to stamp out these types of

special meanings in argument introduction characters.

If you look at the output from `ls -l`, you will see that there are nine white-space separated fields. The file name is in the ninth field, and if you wanted to sort the output into case-independent order using the old arguments to `sort`, you would say

```
$ ls -l | sort -f +8 -9
```

You could read this as "use a sort key starting *after* the eighth field and stopping at the end of the ninth field," i.e., use the ninth field as the primary sort key. Alternatively, you could start counting the fields from zero, and arrive at number eight for the value for the name field. You can choose how to think about the arguments; the effect is the same.



Had we not given a stopping point for the field, then `sort` would use a sorting key commencing from the starting point and terminating at the end of the line. In the example above, the stopping position is not necessary, but I've included it for completeness.

The new specification for `sort` arguments uses the `-k` option followed by a comma-separated range. Also, it starts numbering the fields from one, so

```
$ ls -l | sort -f -k 9,9
```

will start the field at the beginning of the ninth field and finish it at the end of the ninth field. This seems a little more intuitive.

The question then arises about what happens when the fields are identical. This can happen naturally in other data sets and can still be relevant when looking at file names that are guaranteed to be unique. For example, `Fred` is a different file from `fred`, but `sort` will treat the two names as the same when we are using the `-f` option. If you have selected a field and `sort` finds that the fields in two lines are the same, then the default follow-up action is to use the whole line as a sort key to order the two lines.

There are obviously situations where different follow-up actions are needed, and `sort` supplies some further options. The first option is to supply the `-u` argument, suppressing records with identical fields, which allows the `sort` command to take on some of the functionality of the `uniq` command.

If you want to see the replicated data, then the second option is to apply the `sort` modifier just to the field, by putting the magic control letter after the field specification rather than making it apply to all fields by placing it before the `-k` on the command line. We can then provide several different sort criteria for different fields and add multiple field selections:

```
$ ls -l | sort -k 9,9f -k 6,6M
```

Here, we will sort files into case-independent order based on field nine. Then if the names are the same, we will sort based on field six using month name comparison. The `M` modifier sorts the field by recognizing a three-character month name and sorting into month order. Of course, we can go on adding fields to make a better job of sorting on the date in the `ls` output. Incidentally, the `M` modifier is not standard POSIX.

## Field Separators

Field separators have always been a problem for `sort`, and they don't seem to have been completely rationalized by POSIX. POSIX says that in the default case, field separators are treated as part of the sort key. Therefore, spaces and tab characters that will appear to be the same white space on the screen will mysteriously sort into what appears to be a strange order. The folks who wrote the *UNIX Power Tools* book (see Further Reading) came to the conclusion that in the default case you always had one "free" blank character that was not included as part of the field. I suspect that you need to experiment with your implementation to find out what happens, if the separator behavior seems to be causing problems.

One possible light at the end of the separator tunnel is the `-b` option, which tells the `sort` command to fold multiple occurrences of the separator character into one. The general advice in difficult cases where tabs and spaces are used for separators is to replace all the tabs with spaces and use

```
$ sort -t ' ' -b ...
```

to ensure that separators are collapsed to a single character that is not regarded as part of any sorting field.

If you need to specify byte insets into sorting fields, then this is possible with a further piece of syntax. An offset into a field is given by supplying a number after a period in the key definition field, so

```
$ sort -k 2.3b,2.3b
```

will sort the data based on the third nonblank character of the second field. I do suspect that I am now entering realms of the `sort` command that are rarely used, largely because people don't know that the ability to do the job is available, and also because it seems hard to prove that the sorting you need is actually being done by the program.

## Further Reading

As I have mentioned, there's a whole section on sorting in *UNIX Power Tools* by Jerry Peek, Tim O'Reilly and Mike Loukides et al. It's published by O'Reilly & Associates (ISBN 0-553-35402-7).

---

Peter Collinson runs his own UNIX consultancy, dedicated to earning enough money to allow him to pursue his own interests: doing whatever, whenever, wherever… He writes, teaches, consults and programs using Solaris running on a SPARCstation 2. Email: `pc@cpg.com`.