Cut out selected fields of each line of a file

markus schnalke <meillo@marmaro.de>

Cut is a classic program in the Unix toolchest. It is present in most tutorials on shell programming, because it is such a nice and useful tool with good explanatory value. This text shall take a look underneath its surface.

Usage

Initially, cut had two operation modes, which were later amended by a third: The cut program may cut specified characters or bytes out of the input lines or it may cut out specified fields, which are defined by a delimiting character.

The character mode is well suited to slice fixed-width input formats into parts. One might, for instance, extract the access rights from the output of `ls -l`, as shown here with the rights of a file's owner:

```
$ ls -l foo
-rw-rw-r-- 1 meillo users 0 May 12 07:32 foo
$ ls -l foo | cut -c 2-4
rw-
```

Or the write permission for the owner, the group, and the world:

```
$ ls -l foo | cut -c 3,6,9
ww-
```

Cut can also be used to shorten strings:

```
$ long=12345678901234567890
$ echo "$long" | cut -c -10
1234567890
```

This command outputs no more than the first 10 characters of `$long`. (Alternatively, on could use `printf "%.10s\n" "$long"` for this task.)

However, if it's not about displaying characters, but rather about storing them, then `-c` is only partly suited. In former times, when US-ASCII was the omnipresent character encoding, each character was stored as exactly one byte. Therefore, `cut -c` selected both output characters and bytes equally. With the uprise of multi-byte encodings (like UTF-8), this assumption became obsolete. Consequently, a byte mode (option `-b`) was added to cut, with POSIX.2-1992. To select up to 500 bytes from the beginning of each line (and ignore the rest), one can use:

```
$ cut -b -500
```

The remainder can be caught with `cut -b 501-`. This use of cut is important for POSIX, because it provides a transformation of text files with arbitrary line lengths to text files with limited line length [1].

The introduction of the new byte mode essentially held the same functionality as the old character mode. The character mode, however, required a new, different implementation. In consequence, the problem was not the support of the byte mode, but rather the correct support of the new character mode.

Besides the character and byte modes, cut also offers a field mode, which is activated by `-f`. It selects fields from the input. The field-delimiter character for the input as well as for the output (by default the tab) may be changed using `-d`.

The typical example for the use of cut's field mode is the selection of information from the password file. Here, for instance, the usernames and their uids:

```
$ cut -d: -f1,3 /etc/passwd
root:0
bin:1
daemon:2
mail:8
...
```

(The values to the command line switches may be appended directly to them or separated by whitespace.)

The field mode is suited for simple tabular data, like the password file. Beyond that, it soon reaches its limits. The typical case of whitespace-separated fields, in particular, is covered poorly by it. Cut's delimiter is exactly one character, therefore one can not split at both space and tab characters. Furthermore, multiple adjacent delimiter characters lead to empty fields. This is not the expected behavior for the processing of whitespace-separated fields. Some implementations, e.g. the one of FreeBSD, have extensions that handle this case in the expected way. On other systems or to stay portable, awk comes to rescue.

Awk provides another functionality that cut lacks: Changing the order of the fields in the output. For cut, the order of the field selection specification is irrelevant; it doesn't even matter if fields occur multiple times. Thus, the invocation `cut -c 5-8,1,4-6` outputs the characters number 1, 4, 5, 6, 7, and 8 in exactly this order. The selection specification resembles mathematical set theory: Each specified field is part of the solution set. The fields in the solution set are always in the same order as in the input. To speak with the words of the man page in Version 8 Unix: "In data base parlance, it projects a relation." [2] This means that cut applies the *projection* database operation to the text input. Wikipedia explains it in the following way: "In practical terms, it can be roughly thought of as picking a sub-set of all available columns." [3]

Historical Background

Cut came to public life in 1982 with the release of UNIX System III. Browsing through the sources of System III, one finds cut.c with the timestamp 1980-04-11 [4]. This is the oldest implementation of the program I was able to discover. However, the SCCS-ID in the source code contains the version number 1.5. According to Doug McIlroy [5], the earlier history likely lies in PWB/UNIX, which was the basis for System III. In the available sources of PWB 1.0 (1977) [6], no cut is present. Of PWB 2.0, no sources or useful documentation seem to be available. PWB 3.0 was later renamed to System III for marketing purposes only; it is otherwise identical to it. A branch of PWB was CB UNIX, which was only used in the Bell Labs internally. The manual of CB UNIX Edition 2.1 of November 1979 contains the earliest mention of cut that my research brought to light, in the form of a man page [7].

A look at BSD: There, my earliest discovery is a cut.c with the file modification date of 1986-11-07 [8] as part of the special version 4.3BSD-UWisc [9], which was released in January 1987. This implementation is mostly identical to the one in System III. The better known 4.3BSD-Tahoe (1988) does not contain cut. The subsequent 4.3BSD-Reno (1990) does include cut. It is a freshly written one by Adam S. Moskowitz and Marciano Pitargue, which was included in BSD in 1989 [10]. Its man page [11] already mentions the expected compliance to POSIX.2. One should note that POSIX.2 was first published in September 1992, about two years after the man page and the program were written. Hence, the program must have been implemented based on a draft version of the standard. A look into the code confirms the assumption. The function to parse the field selection includes the following comment:

> This parser is less restrictive than the Draft 9 POSIX spec. POSIX doesn't allow lists that aren't in increasing order or overlapping lists.

Draft 11.2 of POSIX (1991-09) requires this flexibility already:

> The elements in list can be repeated, can overlap, and can be specified in any order.

The same draft additionally includes all three operation modes, whereas this early BSD cut only implemented the original two. Draft 9 might not have included the byte mode. Without access to Draft 9 or 10, it wasn't possible to verify this guess.

The version numbers and change dates of the older BSD implementations are manifested in the SCCS-IDs, which the version control system of that time inserted. For instance in 4.3BSD-Reno: "5.3 (Berkeley) 6/24/90".

The cut implementation of the GNU coreutils contains the following copyright notice:

```
Copyright (C) 1997-2015 Free Software Foundation, Inc.
Copyright (C) 1984 David M. Ihnat
```

This code does have old origins. Further comments show that the source code was reworked by David MacKenzie first and later by Jim Meyering, who put it into the version control system in 1992. It is unclear why the years until 1997, at least from 1992 onwards, don't show up in the copyright notice.

Despite all those year numbers from the 80s, cut is a rather young tool, at least in relation to the early Unix. Despite being a decade older than Linux (the kernel), Unix was present for over ten years already by the time cut appeared for the first time. Most notably, cut wasn't part of Version 7 Unix, which became the basis for all modern Unix systems. The more complex tools sed and awk were part of it already. Hence, the question comes to mind why cut was written at all, as two programs already existed that were able to cover its use cases. One reason for cut surely was its compactness and the resulting speed, in comparison to the then-bulky awk. This lean shape goes well with the Unix philosophy: Do one job and do it well! Cut was sufficiently convincing. It found its way to other Unix variants, it became standardized, and today it is present everywhere.

The original variant (without `-b`) was described already in 1985, by the System V Interface Definition, an important formal description of UNIX System V. In the following years, it appeared in all relevant standards. POSIX.2 specified cut for the first time in its modern form (with `-b`) in 1992.

Multi-byte support

The byte mode and thus the multi-byte support of the POSIX character mode have been standardized since 1992. But are they present in the available implementations? Which versions implement POSIX correctly?

The situation is divided into three parts: There are historic implementations, which have only `-c` and `-f`. Then there are implementations that have `-b`, but treat it as an alias for `-c` only. These implementations work correctly for single-byte encodings (e.g. US-ASCII, Latin1) but for multi-byte encodings (e.g. UTF-8) their `-c` behaves like `-b` (and `-n` is ignored). Finally, there are implementations that implement `-c` and `-b` in a POSIX-compliant way.

Historic two-mode implementations are the ones of System III, System V, and the BSD ones until the mid-90s.

Pseudo multi-byte implementations are provided by GNU, modern NetBSD, and modern OpenBSD. The level of POSIX compliance that is presented there is often higher than the level of compliance that is actually provided. Sometimes it takes a close look to discover that `-c` and `-n` don't behave as expected. Some of the implementations take the easy way by simply being ignorant to any multi-byte encodings, at least they declare that clearly:

> Since we don't support multi-byte characters, the `-c` and `-b` options are equivalent, and the `-n` option is meaningless. [12]

Standard-adhering implementations, i.e. ones that treat multi-byte characters correctly, are those of the modern FreeBSD and the Heirloom toolchest. Tim Robbins reimplemented the character mode of FreeBSD cut, conforming to POSIX, in the summer of 2004 [13]. The question why the other BSD systems have not integrated this change is an open one. Maybe the answer is a general ignorance of internationalization.

How do users find out if the cut on their own system handles multi-byte characters correctly? First, one needs to check if the system itself uses multi-byte characters, because otherwise characters and bytes are equivalent and the question is irrelevant. One can check this by looking at the locale

settings, but it is easier to print a typical multi-byte character, for instance an Umlaut or the Euro currency symbol, and check if one or more bytes are generated as output:

```
$ echo ä | od -c
0000000 303 244  \n
0000003
```

In this case it resulted in two bytes: octal 303 and 244. (The newline character is added by echo.)

The program iconv converts text to specific encodings. This is the output for Latin1 and UTF-8, for comparison:

```
$ echo ä | iconv -t latin1 | od -c
0000000 344  \n
0000002
$ echo ä | iconv -t utf8 | od -c
0000000 303 244  \n
0000003
```

The output (without the iconv conversion) on many European systems equals one of these two.

Now for the test of the cut implementation. On a UTF-8 system, a POSIX-compliant implementation behaves as such:

```
$ echo ä | cut -c 1 | od -c
0000000 303 244  \n
0000003
$ echo ä | cut -b 1 | od -c
0000000 303  \n
0000002
$ echo ä | cut -b 1 -n | od -c
0000000  \n
0000001
```

A pseudo-POSIX implementation, in contrast, behaves like the middle one for all three invocations: Only the first byte is printed as output.

Implementations

Let's take a look at the sources of a selection of implementations.

A comparison of the amount of source code is good to get a first impression. Typically, it grows through time. This can generally be seen here, but not in all cases. A POSIX-compliant implementation of the character mode requires more code, thus these implementations tend to be the larger ones.

SLOC	Lines	Bytes	Belongs to	File time	Category
116	123	2966	System III	1980-04-11	historic
118	125	3038	4.3BSD-UWisc	1986-11-07	historic
200	256	5715	4.3BSD-Reno	1990-06-25	historic
200	270	6545	NetBSD	1993-03-21	historic
218	290	6892	OpenBSD	2008-06-27	pseudo-POSIX
224	296	6920	FreeBSD	1994-05-27	historic
232	306	7500	NetBSD	2014-02-03	pseudo-POSIX
340	405	7423	Heirloom	2012-05-20	POSIX
382	586	14175	GNU coreutils	1992-11-08	pseudo-POSIX
391	479	10961	FreeBSD	2012-11-24	POSIX
588	830	23167	GNU coreutils	2015-05-01	pseudo-POSIX

There are four rough groups: (1) The two original implementations, which are mostly identical, with about 100 SLOC. (2) The five BSD versions, with about 200 SLOC. (3) The two POSIX-compliant versions and the old GNU one, with a SLOC count in the 300s. And finally, (4) the modern GNU cut with almost 600 SLOC.

The variation between the number of logical code lines (SLOC, measured with SLOCcount) and the number of newlines in the file (`wc -l`) spans between factor 1.06 for the oldest versions and factor 1.5 for GNU. The largest influence on it are empty lines, pure comment lines, and the size of the license block at the beginning of the file.

Regarding the variation between logical code lines and the file size (`wc -c`), the implementations span between 25 and 30 bytes per statement. With only 21 bytes per statement, the Heirloom implementation marks the lower end; the GNU implementation sets the upper limit at nearly 40 bytes. In the case of GNU, the reason is mainly their coding style, with special indentation rules and long identifiers. Whether one finds the Heirloom implementation [14] highly cryptic or exceptionally elegant shall be left to the judgement of the reader. Especially the comparison to the GNU implementation [15] is impressive.

The internal structure of the source code (in all cases it is written in C) is mainly similar. Besides the mandatory main function, which does the command line argument processing, there usually is a function to convert the field selection specification to an internal data structure. Furthermore, almost all implementations have separate functions for each of their operation modes. The POSIX-compliant versions treat the `-b -n` combination as a separate mode and thus implement it in a separate function. Only the early System III implementation (and its 4.3BSD-UWisc variant) do everything, apart from error handling, in the main function.

Implementations of cut typically have two limiting aspects: One being the maximum number of fields that can be handled, the other being the maximum line length. On System III, both numbers are limited to 512. 4.3BSD-Reno and the BSDs of the 90s have fixed limits as well (`_BSD_LINE_MAX` or `_POSIX2_LINE_MAX`). Modern FreeBSD, modern NetBSD, all GNU implementations, and the Heirloom cut are able to handle arbitrary numbers of fields and line lengths – the memory is allocated dynamically. OpenBSD cut is a hybrid: It has a fixed maximum number of fields, but allows arbitrary line lengths. The limited number of fields does not, however, appear to be any practical problem, because `_POSIX2_LINE_MAX` is guaranteed to be at least 2048 and is thus probably large enough.

Descriptions

Interesting, as well, is a comparison of the short descriptions of cut, as can be found in the headlines of the man pages or at the beginning of the source code files. The following list is roughly grouped by origin:

CB UNIX	cut out selected fields of each line of a file
System III	cut out selected fields of each line of a file
System III †	cut and paste columns of a table (projection of a relation)
System V	cut out selected fields of each line of a file
HP-UX	cut out (extract) selected fields of each line of a file
4.3BSD-UWisc †	cut and paste columns of a table (projection of a relation)
4.3BSD-Reno	select portions of each line of a file
NetBSD	select portions of each line of a file
OpenBSD 4.6	select portions of each line of a file
FreeBSD 1.0	select portions of each line of a file
FreeBSD 10.0	cut out selected portions of each line of a file
SunOS 4.1.3	remove selected fields from each line of a file
SunOS 5.5.1	cut out selected fields of each line of a file
Heirloom Tools	cut out selected fields of each line of a file
Heirloom Tools †	cut out fields of lines of files
GNU coreutils	remove sections from each line of files
Minix	select out columns of a file
Version 8 Unix	rearrange columns of data
“Unix Reader”	rearrange columns of text
POSIX	cut out selected fields of each line of a file

(The descriptions that are marked with '†' were taken from source code files. The POSIX entry contains the description used in the standard. The ''Unix Reader'' is a retrospective document by Doug McIlroy, which lists the availability of tools in the Research Unix versions [16]. Its description should actually match the one in Version 8 Unix. The change could be a transfer mistake or a correction. All other descriptions originate from the various man pages.)

Over time, the POSIX description was often adopted or it served as inspiration. One such example is FreeBSD [17].

It is noteworthy that the GNU coreutils in all versions describe the performed action as a removal of parts of the input, although the user clearly selects the parts that then consistute the output. Probably the words ''cut out'' are too misleading. HP-UX tried to be more clear.

Different terms are also used for the part being selected. Some talk about fields (POSIX), some talk about portions (BSD) and some call it columns (Research Unix).

The seemingly least adequate description, the one of Version 8 Unix (''rearrange columns of data'') is explainable in so far as that the man page covers both cut and paste, and in their combination, columns can be rearranged. The use of the word ''data'' instead of ''text'' might be a lapse, which McIlroy corrected in his Unix Reader ... but on the other hand, on Unix, the two words are mostly synonymous, because all data is text.

References

1 http://pubs.opengroup.org/onlinepubs/9699919799/utilities/cut.html#tag_20_28_17
2 http://man.cat-v.org/unix_8th/1/cut
3 https://en.wikipedia.org/wiki/Projection_(relational_algebra)
4 http://minnie.tuhs.org/cgi-bin/utree.pl?file=SysIII/usr/src/cmd
5 http://minnie.tuhs.org/pipermail/tuhs/2015-May/004083.html
6 http://minnie.tuhs.org/Archive/PDP-11/Distributions/usdl/
7 ftp://sunsite.icm.edu.pl/pub/unix/UnixArchive/PDP-11/Distributions/other/CB_Unix/cbunix_man1_02.pdf
8 http://minnie.tuhs.org/cgi-bin/utree.pl?file=4.3BSD-UWisc/src/usr.bin/cut
9 http://gunkies.org/wiki/4.3_BSD_NFS_Wisconsin_Unix
10 http://minnie.tuhs.org/cgi-bin/utree.pl?file=4.3BSD-Reno/src/usr.bin/cut
11 http://minnie.tuhs.org/cgi-bin/utree.pl?file=4.3BSD-Reno/src/usr.bin/cut/cut.1
12 http://cvsweb.openbsd.org/cgi-bin/cvsweb/src/usr.bin/cut/cut.c?rev=1.18&content-type=text/x-cvsweb-markup
13 https://svnweb.freebsd.org/base?view=revision&revision=131194
14 http://heirloom.cvs.sourceforge.net/viewvc/heirloom/heirloom/cut/cut.c?revision=1.6&view=markup
15 http://git.savannah.gnu.org/gitweb/?p=coreutils.git;a=blob;f=src/cut.c;hb=e981643
16 http://doc.cat-v.org/unix/unix-reader/contents.pdf
17 https://svnweb.freebsd.org/base?view=revision&revision=167101