cut out selected fields of each line of a file

von Markus Schnalke

Ein klassisches Programm im Unix-Werkzeugkasten ist cut. In keinem ordentlichen Tutorial zur Shellprogrammierung fehlt es, denn es ist ein schönes, praktisches und anschauliches Helferlein. Hier soll ein wenig hinter seine Fassade geschaut werden.

Funktionsweise

Ursprünglich hatte cut zwei Modi, die später um einen dritten erweitert wurden. cut schneidet entweder gewünschte Zeichen aus den Zeilen der Eingabe oder gewünschte, durch Trennzeichen definierte, Felder.

Der Zeichenmodus ist optimal geeignet, um Festbreitenformate zu zerteilen. Man kann damit beispielsweise bestimmte Zugriffsrechte aus der Ausgabe von **`ls -l`** ausschneiden, in diesem Beispiel die Rechte des Besitzers:

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

Oder die Schreibrechte des Besitzers, der Gruppe und der Welt:

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

Mit cut lassen sich aber auch Strings kürzen:

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

Dieser Befehl gibt die ersten maximal 10 Zeichen von **`$long`** aus. (Alternativ kann man hierfür **`printf "%.10s\n" "$long"`** verwenden.)

Geht es aber nicht um die Darstellung von Zeichen, sondern um ihre Speicherung, dann ist **`-c`** nicht unbedingt geeignet. Früher, als US-ASCII noch die omnipräsente Zeichenkodierung war, wurde jedes Zeichen mit genau einem Byte gespeichert. Somit selektierte **`cut -c`** gleichermaßen sowohl Ausgabezeichen als auch Bytes. Mit dem Aufkommen von Multibyte-Kodierungen (wie UTF-8) musste man sich jedoch von dieser Annahme lösen. In diesem Zug bekam cut mit POSIX.2-1992 einen Bytemodus (Option **`-b`**). Will man also nur die ersten maximal 500 Bytes vor dem Newline-Zeichen stehen haben (und den Rest stillschweigend ignorieren), dann macht man das mit:

```
cut -b -500
```

Den Rest kann man sich mit **`cut -b 501-`** einfangen. Diese Funktion ist insbesondere für POSIX wichtig, da man damit Textdateien mit begrenzter Zeilenlänge erzeugen kann [1].

Wenn auch der Bytemodus neu eingeführt worden war, so sollte er sich doch nur so verhalten wie der alte Zeichenmodus normalerweise schon implementiert war. Beim Zeichenmodus aber wurde eine neue Implementierungsweise gefordert. Das Problem war folglich nicht, den neuen Bytemodus zu implementieren, sondern den Zeichenmodus neu zu implementieren.

Neben dem Zeichen- und Bytemodus bietet cut noch den Feldmodus, den man mit **`-f`** einleitet. Mit ihm ist es möglich, Felder auszuwählen. Das Trennzeichen (per Default der Tab) kann mit **`-d`** geändert werden. Es gilt in gleicher Weise für die Eingabe und die Ausgabe.

Der typische Anwendungsfall für cut im Feldmodus ist die Auswahl von Information aus der **`passwd`**-Datei. Hier z. B. der Benutzername und seine ID:

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

Die einzelnen Argumente für die Optionen können bei cut übrigens sowohl mit Whitespace abgetrennt (wie oben zu sehen) als auch direkt angehängt folgen.

Dieser Feldmodus ist für einfache tabellarische Dateien, wie eben die **passwd**-Datei, gut geeignet. Er kommt aber schnell an seine Grenzen. Gerade der häufige Fall, dass an Whitespace in Felder geteilt werden soll, wird damit nicht abgedeckt. Der Delimiter kann bei cut nur genau ein Zeichen sein. Es kann demnach nicht sowohl an Leerzeichen als auch an Tabs aufgetrennt werden. Zudem unterteilt cut an jedem Trennzeichen. Zwei aneinander stehende Trennzeichen führen zu einem leeren Feld. Dieses Verhalten widerspricht den Erwartungen, die man an die Verarbeitung einer Datei mit Whitespace-getrennten Feldern hat. Manche Implementierungen von cut, z. B. die von FreeBSD, haben deshalb Erweiterungen, die das gewünschte Verhalten für Whitespace-getrennte Felder bieten. Ansonsten, d. h. wenn man portabel bleiben will, verwendet man awk in diesen Fällen.

awk bietet noch eine weitere Funktion, die cut missen lässt: Das Tauschen der Feld-Reihenfolge in der Ausgabe. Bei cut ist die Reihenfolge der Feldauswahlangabe irrelevant; ein Feld kann selbst mehrfach angegeben werden. Dementsprechend gibt der Aufruf von `cut -c 5-8,1,4-6` die Zeichen Nummer 1, 4, 5, 6, 7 und 8 in genau dieser Reihenfolge aus. Die Auswahl entspricht damit der Mengenlehre in der Mathematik: Jedes angegebene Feld wird Teil der Ergebnismenge. Die Felder der Ergebnismenge sind hierbei immer gleich geordnet wie in der Eingabe. Um die Worte der Manpage von Version 8 Unix wiederzugeben: „*In data base parlance, it projects a relation.*“ [2]

Cut führt demnach die Datenbankoperation Projektion auf Textdateien aus. Die Wikipedia erklärt das folgendermaßen [3]:

„*Die Projektion entspricht der Projektionsabbildung aus der Mengenlehre und kann auch Attributbeschränkung genannt werden. Sie extrahiert einzelne Attribute aus der ursprünglichen Attributmenge und ist somit als eine Art Selektion auf Spaltenebene zu verstehen, das heißt, die Projektion blendet Spalten aus.*“

Geschichtliches

Cut erblickte 1982 mit dem Release von UNIX System III das Licht der öffentlichen Welt. Wenn man die Quellen von System III durchforstet, findet man **cut.c** mit dem Zeitstempel 1980-04-11 [4]. Das ist die älteste Implementierung des Programms, die ich aufstöbern konnte. Allerdings spricht die SCCS-ID im Quellcode von Version 1.5. Die Vorgeschichte liegt – der Vermutung Doug McIlroys [5] zufolge – in PWB/UNIX, dessen Entwicklungslinie die Grundlage für System III war. In den von PWB 1.0 (1977) verfügbaren Quellen [6] ist cut noch nicht zu finden. Von PWB 2.0 scheinen keine Quellen oder hilfreiche Dokumentation verfügbar zu sein. PWB 3.0 wurde später aus Marketinggründen als System III bezeichnet und ist folglich mit ihm identisch. Eine Nebenlinie zu PWB war CB UNIX, das nur innerhalb der Bell Labs genutzt wurde. Das Handbuch von CB UNIX Edition 2.1 vom November 1979 enthält die früheste Erwähnung von cut, die meine Recherche zutage gefördert hat: eine Manpage für cut [7].

Nun ein Blick auf die BSD-Linie: Dort ist der früheste Fund ein **cut.c** mit dem Dateimodifikationsdatum 1986-11-07 [8] als Teil der Spezialversion 4.3BSD-UWisc [9], die im Januar 1987 veröffentlicht wurde. Die Implementierung unterscheidet sich nur minimal von der in System III. Im bekannteren 4.3BSD-Tahoe (1988) tauchte cut nicht auf. Das darauf folgende 4.3BSD-Reno (1990) enthielt aber wieder ein cut. Dieses cut war ein von Adam S. Moskowitz und Marciano Pitargue neu implementiertes cut, das 1989 in BSD aufgenommen wurde [10]. Seine Manpage [11] erwähnt bereits die erwartete Konformität mit POSIX.2. Nun muss man wissen, dass POSIX.2 erst im September 1992 veröffentlicht wurde, also erst gut zwei Jahre, nachdem Manpage und Programm geschrieben worden waren. Das Programm wurde folglich anhand von Arbeitsversionen des Standards implementiert. Ein Blick in den Code bekräftigt diese Vermutung. In der Funktion zum Parsen der Feldauswahlliste findet sich dieser Kommentar:

„*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.*“

Im Draft 11.2 (1991-09) fordert POSIX diese Flexibilität bereits ein:

„*The elements in list can be repeated, can overlap, and can be specified in any order.*“

Zudem listet Draft 11.2 alle drei Modi, während in diesem BSD cut nur die zwei alten implementiert

sind. Es könnte also sein, dass in Draft 9 der Bytemodus noch nicht vorhanden war. Ohne Zugang zu Draft 9 oder 10 war es leider nicht möglich, diese Vermutung zu prüfen.

Die Versionsnummern und Änderungsdaten der älteren BSD-Implementierungen kann man aus den SCCS-IDs, die vom damaligen Versionskontrollsystem in den Code eingefügt wurden, ablesen. So z. B. bei 4.3BSD-Reno: „5.3 (Berkeley) 6/24/90“.

Das cut der GNU Coreutils enthält folgenden Copyrightvermerk:

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

Der Code hat also recht alte Ursprünge. Wie aus weiteren Kommentaren zu entnehmen ist, wurde der Programmcode zuerst von David MacKenzie und später von Jim Meyering überarbeitet. Letzterer hat den Code 1992 auch ins Versionskontrollsystem eingestellt. Weshalb die Jahre vor 1997, zumindest ab 1992, nicht im Copyright-Vermerk auftauchen, ist unklar.

Trotz der vielen Jahreszahlen aus den 80er Jahren gehört cut, aus Sicht des ursprünglichen Unix, zu den jüngeren Tools. Wenn cut auch ein Jahrzehnt älter als Linux, der Kernel, ist, so war Unix schon über zehn Jahre alt, als cut das erste Mal auftauchte. Insbesondere gehörte cut noch nicht zu Version 7 Unix, das die Ausgangsbasis aller modernen Unix-Systeme darstellt. Die weit komplexeren Programme sed und awk waren dort aber schon vertreten. Man muss sich also fragen, warum cut überhaupt noch entwickelt wurde, wo es schon zwei Programme gab, die die Funktion von cut abdecken konnten. Ein Argument für cut war sicher seine Kompaktheit und die damit verbundene Geschwindigkeit gegenüber dem damals trägen awk. Diese schlanke Gestalt ist es auch, die der Unix-Philosophie entspricht: Mache eine Aufgabe und die richtig! cut überzeugte. Es wurde in andere Unix-Varianten übernommen, standardisiert und ist heutzutage überall anzutreffen.

Die ursprüngliche Variante (ohne **-b**) wurde schon 1985 in der System V Interface Definition, einer wichtigen formalen Beschreibung von UNIX System V, spezifiziert und tauchte anschließend in allen relevanten Standards auf. Mit POSIX.2 im Jahre 1992 wurde cut zum ersten Mal in der heutigen Form (mit **-b**) standardisiert.

Multibyte-Unterstützung

Nun sind der Bytemodus und die damit verbundene Multibyte-Verarbeitung des POSIX-Zeichenmodus bereits seit 1992 standardisiert, wie steht es aber mit deren Umsetzung? Welche Versionen implementieren POSIX korrekt? Die Situation ist dreiteilig: Es gibt historische Implementierungen, die nur **-c** und **-f** kennen. Dann gibt es Implementierungen, die **-b** zwar kennen, es aber lediglich als Alias für **-c** handhaben. Diese Implementierungen funktionieren mit Single-Byte-Encodings (z. B. US-ASCII, Latin1) korrekt, bei Multibyte-Encodings (z. B. UTF-8) verhält sich ihr **-c** aber wie **-b** (und **-n** wird ignoriert). Schließlich gibt es noch Implementierungen, die **-b** und **-c** tatsächlich POSIX-konform implementieren.

Historische Zwei-Modi-Implementierungen sind z. B. die von System III, System V und die aller BSDs bis in die 90er.

Pseudo-Multibyte-Implementierungen bieten GNU und die modernen NetBSDs und OpenBSDs. Man darf sich sicher fragen, ob dort ein Schein von POSIX-Konformität gewahrt wird. Teilweise findet man erst nach genauerer Suche heraus, dass **-c** und **-n** nicht wie erwartet funktionieren; teilweise machen es sich die Systeme auch einfach, indem sie auf Singlebyte-Zeichenkodierungen beharren, das aber dafür klar darlegen [12]:

„*Since we don't support multi-byte characters, the -c and -b options are equivalent, and the -n option is meaningless.*“

Tatsächlich standardkonforme Implementierungen, die Multibytes korrekt handhaben, bekommt man bei einem modernen FreeBSD und bei den Heirloom Tools. Bei FreeBSD hat Tim Robbins im Sommer 2004 den Zeichenmodus POSIX-konform reimplementiert [13]. Warum die beiden anderen großen BSDs diese Änderung nicht übernommen haben, bleibt offen. Es scheint aber an

der im obigen Kommentar formulierten Grundausrichtung zu liegen.

Wie findet man nun als Nutzer heraus, ob beim cut des eigenen Systems Multibytes korrekt unterstützt werden? Zunächst einmal ist entscheidend, ob das System selbst mit einem Multibyte-Encoding arbeitet, denn tut es das nicht, dann entsprechen sich Zeichen und Bytes und die Frage erübrigt sich. Man kann das herausfinden indem man sich das Locale anschaut, aber einfacher ist es, ein typisches Mehrbytezeichen, wie zum Beispiel einen Umlaut, auszugeben und zu schauen ob dieses in einem oder in mehreren Bytes kodiert ist:

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

In diesem Fall sind es zwei Bytes: oktal 303 und 244. (Den Zeilenumbruch fügt echo hinzu.)

Mit dem Programm iconv kann man Text explizit in bestimmte Kodierungen konvertieren. Hier Beispiele, wie die Ausgabe bei Latin1 und wie sie bei UTF-8 aussieht:

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

Die Ausgabe auf dem eigenen System (ohne die iconv-Konvertierung) wird recht sicher einer dieser beiden Ausgaben entsprechen.

Nun zum Test der cut-Implementierung. Hat man ein UTF-8-System, dann sollte sich eine POSIX-konforme Implementierung folgendermaßen verhalten:

```
$ 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
```

Bei einer Pseudo-POSIX-Implementierung ist die Ausgabe in allen drei Fällen wie die mittlere: Es wird das erste Byte ausgegeben.

Implementierungen

Nun ein Blick auf den Code. Betrachtet wird eine Auswahl an Implementierungen.

Für einen ersten Eindruck ist der Umfang des Quellcodes hilfreich. Typischerweise steigt dieser über die Jahre an. Diese Beobachtung kann hier in der Tendenz, aber nicht in jedem Fall, bestätigt werden. Die POSIX-konforme Umsetzung des Zeichenmodus erfordert zwangsläufig mehr Code, deshalb sind diese Implementierungen tendenziell umfangreicher.

In der Tabelle mit „(hist)“ bezeichnete Implementierungen beziehen sich auf ältere Implementierungen, die nur **`-c`** und **`-f`** kennen. Pseudo-Implementierungen, die **`-b`** zwar kennen, es aber nur als Alias für **`-c`** handhaben, sind mit „(pseudo)“ gekennzeichnet. POSIX-konforme Implementierungen von cut sind mit „(POSIX)“ markiert.

Das Kandidatenfeld teilt sich grob in vier Gruppen:

1. Die zwei ursprünglichen Implementierungen, die sich nur minimal unterscheiden, mit gut 100 SLOCs.
2. Die fünf BSD-Versionen mit gut 200 SLOCs.
3. Die zwei POSIX-konformen Programme und die alte GNU-Version mit 340-390 SLOCs.
4. Und schließlich die moderne GNU-Variante mit fast 600 SLOCs.

Die Abweichung zwischen logischen Codezeilen (SLOC, ermittelt mit SLOCcount) und der Anzahl von Zeilenumbrüchen in der Datei (**`wc -l`**) erstreckt sich über eine Spanne von Faktor 1.06 bei den ältesten Vertretern bis zu Faktor 1.5 bei GNU. Den größten Einfluss darauf haben Leerzeilen, reine Kommentarzeilen und die Größe des Lizenzblocks am Dateianfang.

Betrachtet man die Abweichungen zwischen den logischen Codezeilen und der Dateigröße (**`wc -c`**), so pendelt das Teilnehmerfeld zwischen

Verschiedene Implementierungen von cut

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

25 und 30 Bytes je Anweisung. Die Heirloom-Implementierung weicht mit nur 21 nach unten ab, die GNU-Implementierungen mit fast 40 nach oben. Bei GNU liegt dies hauptsächlich an deren Programmierstil, mit spezieller Einrückung und langen Bezeichnern. Ob man die Heirloom-Implementierung [14] als besonders kryptisch oder als besonders elegant bezeichnen will, das soll der eigenen Einschätzung des Lesers überlassen bleiben. Vor allem der Vergleich mit einer GNU-Implementierung [15] ist eindrucksvoll.

Die interne Struktur der Programmcodes (in C) ist meist ähnlich. Neben der obligatorischen main-Funktion, die die Kommandozeilenargumente verarbeitet, gibt es im Normalfall eine Funktion, die die Feldauswahl in eine interne Datenstruktur überführt. Desweiteren haben fast alle Implementierungen separate Funktionen für jeden ihrer Modi. Bei den POSIX-konformen Implementierungen wird die Kombination `-b -n` als weiterer Modus behandelt, und damit in einer eigenen Funktion umgesetzt. Nur bei der frühen System III-Implementierung (und seiner 4.3BSD-UWisc-Variante) wird außer den Fehlerausgaben alles in der main-Funktion erledigt.

cut-Implementierungen haben typischerweise zwei limitierende Größen: Die Maximalanzahl unterstützter Felder und die maximale Zeilenlänge. Bei System III sind beide Größen auf 512 begrenzt. 4.3BSD-Reno und die BSDs der 90er Jahre haben ebenfalls fixe Grenzen (`_BSD_LINE_MAX` bzw. `_POSIX2_LINE_MAX`). Bei modernen FreeBSDs, NetBSDs, bei allen GNU-Implementierungen und bei Heirloom kann sowohl die Felderanzahl als auch die maximale Zeilenlänge beliebig groß werden; der Speicher dafür wird dynamisch alloziert. OpenBSD ist ein Hybrid aus fixer Maximalzahl an Feldern, aber beliebiger Zeilenlänge. Die begrenzte Felderanzahl scheint jedoch kein Praxisproblem darzustellen, da `_POSIX2_LINE_MAX` mit mindestens 2048 durchaus groß genug sein sollte.

Beschreibungen

Interessant ist zudem ein Vergleich der Kurzbeschreibungen von cut, wie sie sich in der Titelzeile der Manpages oder manchmal am Anfang der Quellcodedatei finden. Die folgende Liste ist grob zeitlich geordnet und nach Abstammung gruppiert:

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 (src): „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 (src): „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 (src): „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“

Die mit „(src)“ markierten Beschreibungen sind aus dem jeweiligen Quellcode entnommen. Der POSIX-Eintrag enthält die Beschreibung im Standard. Der „Unix Reader“ ist ein rückblickendes Textdokument von Doug McIlroy, das das Auftreten der Tools in der Geschichte des Research Unix zum Thema hat [16]. Eigentlich sollte seine Beschreibung der in Version 8 Unix entsprechen. Die Abweichung könnte ein Übertragungsfehler oder eine nachträgliche Korrektur sein. Alle übrigen Beschreibungen entstammen den Manpages.

Oft ist mit der Zeit die POSIX-Beschreibung übernommen oder an sie angeglichen worden, wie beispielsweise bei FreeBSD [17].

Interessant ist, dass die GNU coreutils seit Anbeginn vom Entfernen von Teilen der Eingabe sprechen, wohingegen die Kommandozeilenangabe klar ein Auswählen darstellt. Die Worte „cut out“ sind vielleicht auch zu missverständlich. HP-UX hat sie deshalb präzisiert.

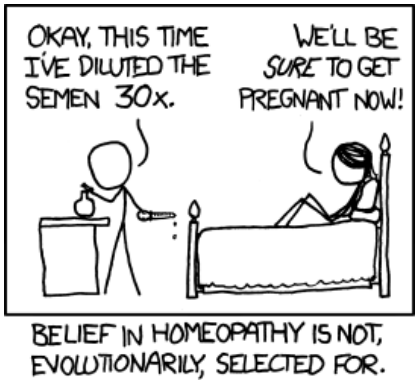
Beim Begriff, was selektiert wird, ist man sich ebenfalls uneins. Es wird von Feldern (POSIX), Abschnitten bzw. Teilen (BSD) oder Spalten (Research Unix) geredet. Die seltsame Beschreibung bei Version 8 Unix („rearrange columns of data“) ist dadurch zu erklären, dass sie aus der gemeinsamen Manpage für cut und paste stammt. In der Kombination der beiden Werkzeuge können nämlich Spalten umgeordnet werden.

Links

[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://de.wikipedia.org/wiki/Projektion_(Informatik)#Projektion
[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

Autoreninformation

Markus Schnalke interessiert sich für die Hintergründe von Unix und seinen Werkzeugen. Für die Erarbeitung dieses Textes wurde er regelrecht zum Historiker.



“Dilution” © by Randall Munroe (CC-BY-NC-2.5), http://xkcd.com/765/