

hexdump — Display hexadecimal report on file

Description Remarks and examples	Syntax Stored results	Options Also see
---	--	---

Description

`hexdump` displays a hexadecimal dump of a file or, optionally, a report analyzing the dump.

Syntax

```
hexdump filename [ , options ]
```

<i>options</i>	Description
<u>a</u> nalyze	display a report on the dump rather than the dump itself
<u>t</u> abulate	display a full tabulation of the ASCII and extended ASCII characters in the <code>analyze</code> report
<u>n</u> oextended	do not display printable extended ASCII characters
<u>r</u> esults	store results containing the frequency with which each character code was observed; programmer's option
<u>f</u> rom(#)	dump or analyze first byte of the file; default is to start at first byte, <code>from(0)</code>
<u>t</u> o(#)	dump or analyze last byte of the file; default is to continue to the end of the file

Options

`analyze` specifies that a report on the dump, rather than the dump itself, be presented.

`tabulate` specifies in the `analyze` report that a full tabulation of the ASCII and extended ASCII characters also be presented.

`noextended` specifies that `hexdump` not display printable extended ASCII characters, characters in the range 161–254 or, equivalently, 0xa1–0xfe. (`hexdump` does not display characters 128–160 and 255.)

`results` is for programmers. It specifies that, in addition to other stored results, `hexdump` store `r(c0)`, `r(c1)`, ..., `r(c255)`, containing the frequency with which each character code was observed.

`from(#)` specifies the first byte of the file to be dumped or analyzed. The default is to start at the first byte of the file, `from(0)`.

`to(#)` specifies the last byte of the file to be dumped or analyzed. The default is to continue to the end of the file.

Remarks and examples

`hexdump` is useful when you are having difficulty reading a file with `infile`, `infix`, or `import delimited`. Sometimes, the reason for the difficulty is that the file does not contain what you think it contains, or that it does contain the format you have been told, and looking at the file in text mode is either not possible or not revealing enough.

Pretend that we have the file `myfile.raw` containing

```
Datsun 210      4589 35 5 1
VW Scirocco    6850 25 4 1
Merc. Bobcat   3829 22 4 0
Buick Regal    5189 20 3 0
VW Diesel      5397 41 5 1
Pont. Phoenix  4424 19 . 0
Merc. Zephyr   3291 20 3 0
Olds Starfire  4195 24 1 0
BMW 320i       9735 25 4 1
```

We will use `myfile.raw` with `hexdump` to produce output that looks like the following:

```
. hexdump myfile.raw
```

address	hex representation										character representation	
	0 1	2 3	4 5	6 7	8 9	a b	c d	e f	0123456789abcdef			
0	4461	7473	756e	2032	3130	2020	2020	2034			Datsun 210	4
10	3538	3920	2033	3520	2035	2020	310a	5657			589 35 5 1.VW	
20	2053	6369	726f	6363	6f20	2020	2036	3835			Scirocco 685	
30	3020	2032	3520	2034	2020	310a	4d65	7263			0 25 4 1.Merc	
40	2e20	426f	6263	6174	2020	2033	3832	3920			. Bobcat 3829	
50	2032	3220	2034	2020	300a	4275	6963	6b20			22 4 0.Buick	
60	5265	6761	6c20	2020	2035	3138	3920	2032			Regal 5189 2	
70	3020	2033	2020	300a	5657	2044	6965	7365			0 3 0.VW Diese	
80	6c20	2020	2020	2035	3339	3720	2034	3120			1 5397 41	
90	2035	2020	310a	506f	6e74	2e20	5068	6f65			5 1.Pont. Phoe	
a0	6e69	7820	2034	3432	3420	2031	3920	202e			nix 4424 19 .	
b0	2020	300a	4d65	7263	2e20	5a65	7068	7972			0.Merc. Zephyr	
c0	2020	2033	3239	3120	2032	3020	2033	2020			3291 20 3	
d0	300a	4f6c	6473	2053	7461	7266	6972	6520			0.Olds Starfire	
e0	2034	3139	3520	2032	3420	2031	2020	300a			4195 24 1 0.	
f0	424d	5720	3332	3069	2020	2020	2020	2039			BMW 320i 9	
100	3733	3520	2032	3520	2034	2020	310a				735 25 4 1.	

hexdump can also produce output that looks like the following:

```
. hexdump myfile.raw, analyze
Line-end characters          Line length (tab=1)
  \r\n      (Windows)      0      minimum          29
  \r by itself (Mac)      0      maximum           29
  \n by itself (Unix)     9
Space/separator characters  Number of lines          9
 [blank]          99      EOL at EOF?          yes
 [tab]            0
 [comma] (,)      0      Length of first 5 lines
Control characters          Line 1          29
 binary 0        0      Line 2          29
 CTL excl. \r, \n, \t  0      Line 3          29
 DEL             0      Line 4          29
 Extended (128-159,255)  0      Line 5          29
ASCII printable
 A-Z             20
 a-z             61      File format          ASCII
 0-9             77
 Special (!@#$ etc.)    4
 Extended (160-254)     0
-----
Total                270
Observed were:
 \n blank . 0 1 2 3 4 5 6 7 8 9 B D M 0 P R S V W Z a b c d e f g h i k l
 n o p r s t u x y
```

Of the two forms of output, the second is often the more useful because it summarizes the file, and the length of the summary is not a function of the length of the file. Here is the summary for a file that is just over 4 MB long:

```
. hexdump bigfile.raw, analyze
Line-end characters          Line length (tab=1)
  \r\n      (Windows)    147,456      minimum          29
  \r by itself (Mac)      0      maximum          30
  \n by itself (Unix)     2
Space/separator characters  Number of lines    147,458
 [blank]          1,622,039      EOL at EOF?          yes
 [tab]            0
 [comma] (,)      0      Length of first 5 lines
Control characters          Line 1          30
 binary 0        0      Line 2          30
 CTL excl. \r, \n, \t  0      Line 3          30
 DEL             0      Line 4          30
 Extended (128-159,255)  0      Line 5          30
ASCII printable
 A-Z             327,684
 a-z             999,436      File format          ASCII
 0-9             1,261,587
 Special (!@#$ etc.)    65,536
 Extended (160-254)     0
-----
Total                4,571,196
Observed were:
 \n \r blank . 0 1 2 3 4 5 6 7 8 9 B D M 0 P R S V W Z a b c d e f g h i
 k l n o p r s t u x y
```

4 hexdump — Display hexadecimal report on file

Here is the same file but with a subtle problem:

```
. hexdump badfile.raw, analyze
Line-end characters                               Line length (tab=1)
  \r\n      (Windows)                          147,456      minimum                30
  \r by itself (Mac)                            0           maximum                90
  \n by itself (Unix)                           0
Space/separator characters                       Number of lines      147,456
 [blank]                                         1,622,016      EOL at EOF?          yes
 [tab]                                           0
 [comma] (,)                                    0           Length of first 5 lines
Control characters                               Line 1              30
 binary 0                                       8           Line 2              30
 CTL excl. \r, \n, \t                          4           Line 3              30
 DEL                                             0           Line 4              30
 Extended (128-159,255)                        24          Line 5              30
ASCII printable
 A-Z                                           327,683
 a-z                                           999,426      File format          BINARY
 0-9                                           1,261,568
 Special (!@#$ etc.)                          65,539
 Extended (160-254)                            16
-----
Total                                           4,571,196
Observed were:
 \0 ^C ^D ^G \n \r ^U blank & . 0 1 2 3 4 5 6 7 8 9 B D E M O P R S U V W
 Z a b c d e f g h i k l n o p r s t u v x y } ~ E^A E^C E^I E^M E^P
 è é ö 255
```

In the above, the line length varies between 30 and 90 (we were told that each line would be 30 characters long). Also the file contains what `hexdump, analyze` labeled control characters. Finally, `hexdump, analyze` declared the file to be BINARY rather than ASCII.

We created the second file by removing two valid lines from `bigfile.raw` (60 characters) and substituting 60 characters of binary junk. We would defy you to find the problem without using `hexdump, analyze`. You would succeed, but only after much work. Remember, this file has 147,456 lines, and only two of them are bad. If you print 1,000 lines at random from the file, your chances of listing the bad part are only 0.013472. To have a 50% chance of finding the bad lines, you would have to list 52,000 lines, which is to say, review about 945 pages of output. On those 945 pages, each line would need to be drawn at random. More likely, you would list lines in groups, and that would greatly reduce your chances of encountering the bad lines.

The situation is not as dire as we make it out to be because, were you to read `badfile.raw` by using `infile`, it would complain, and here it would tell you exactly where it was complaining. Still, at that point you might wonder whether the problem was with how you were using `infile` or with the data. Moreover, our 60 bytes of binary junk experiment corresponds to transmission error. If the problem were instead that the person who constructed the file constructed two of the lines differently, `infile` might not complain, but later you would notice some odd values in your data (because obviously you would review the summary statistics, right?). Here `hexdump, analyze` might be the only way you could prove to yourself and others that the raw data need to be reconstructed.

Stored results

`hexdump`, `analyze` and `hexdump`, results store the following in `r()`:

Scalars

<code>r(Windows)</code>	number of <code>\r\n</code>
<code>r(Mac)</code>	number of <code>\r</code> by itself
<code>r(Unix)</code>	number of <code>\n</code> by itself
<code>r(blank)</code>	number of blanks
<code>r(tab)</code>	number of tab characters
<code>r(comma)</code>	number of comma (,) characters
<code>r(ctl)</code>	number of binary 0s; A–Z, excluding <code>\r</code> , <code>\n</code> , <code>\t</code> ; DELs; and 128–159, 255
<code>r(uc)</code>	number of A–Z
<code>r(lc)</code>	number of a–z
<code>r(digit)</code>	number of 0–9
<code>r(special)</code>	number of printable special characters (!@#, etc.)
<code>r(extended)</code>	number of printable extended characters (160–254)
<code>r(filesize)</code>	number of characters
<code>r(lmin)</code>	minimum line length
<code>r(lmax)</code>	maximum line length
<code>r(lnum)</code>	number of lines
<code>r(eoleof)</code>	1 if EOL at EOF, 0 otherwise
<code>r(l1)</code>	length of 1st line
<code>r(l2)</code>	length of 2nd line
<code>r(l3)</code>	length of 3rd line
<code>r(l4)</code>	length of 4th line
<code>r(l5)</code>	length of 5th line
<code>r(c0)</code>	number of binary 0s (results only)
<code>r(c1)</code>	number of binary 1s (^A) (results only)
<code>r(c2)</code>	number of binary 2s (^B) (results only)
...	...
<code>r(c255)</code>	number of binary 255s (results only)

Macros

<code>r(format)</code>	ASCII, EXTENDED ASCII, or BINARY
------------------------	----------------------------------

Also see

[D] [filefilter](#) — Convert ASCII or binary patterns in a file

[D] [type](#) — Display contents of a file

Stata, Stata Press, and Mata are registered trademarks of StataCorp LLC. Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations. StataNow and NetCourseNow are trademarks of StataCorp LLC. Other brand and product names are registered trademarks or trademarks of their respective companies. Copyright © 1985–2023 StataCorp LLC, College Station, TX, USA. All rights reserved.



For suggested citations, see the FAQ on [citing Stata documentation](#).