hexdump — Display hexadecimal report on file

Description

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

Syntax

hexdump *filename* [, *options*]

options	Description
<u>analyze</u>	display a report on the dump rather than the dump itself
<u>tab</u>ulate	display a full tabulation of the ASCII and extended ASCII characters in the analyze report
<u>noe</u>xtended	do not display printable extended ASCII characters
<u>res</u>ults	store results containing the frequency with which each character code was observed; programmer's option
<u>from</u>(#)	dump or analyze first byte of the file; default is to start at first byte, from(0)
<u>to</u>(#)	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 $r(c0)$, $r(c1), \ldots, r(c255)$, 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
                                                     character
                          hex representation       representation
       address  |  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 | l       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 O 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 O 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
```

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
  ë 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.

❑ Technical note

In the full hexadecimal dump,

```
. hexdump myfile.raw
                                                    character
                             hex representation     representation
         address    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 310d 0a56  589  35   5  1..V
              20  | 5720 5363 6972 6f63 636f 2020 2020 3638  W Scirocco     68
              30  | 3530 2020 3235 2020 3420 2031 0d0a 4d65  50   25   4  1..Me
```
(output omitted)

addresses (listed on the left) are listed in hexadecimal. Above, 10 means decimal 16, 20 means decimal 32, and so on. Sixteen characters are listed across each line.

In some other dump, you might see something like

```
. hexdump myfile2.raw
                                                    character
                             hex representation     representation
         address    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 3120 2020  589  35   5  1
              20  | 2020 2020 2020 2020 2020 2020 2020 2020
               *
             160  | 2020 2020 2020 0a56 5720 5363 6972 6f63            .VW Sciroc
             170  | 636f 2020 2020 3638 3530 2020 3235 2020  co     6850  25
```
(output omitted)

The ∗ in the address field indicates that the previous line is repeated until we get to hexadecimal address 160 (decimal 352).

❑

Stored results

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

Scalars

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

Macros

r(format)	ASCII, EXTENDED ASCII, or BINARY

Also see

[D] **filefilter** — Convert ASCII or binary patterns in a file

[D] **type** — Display contents of a file