DD - команда, которая не похожа на другие

Экспорт в PDF

Дата создания: 2022/06/20 09:12 (C) mihanik

Немного истории

Команда dd практически ровесник ОС Юникс. Днем рождения последней считается 1 января 1970, и точно известно, что уже в 1970 году утилита dd работала с ленточными накопителями, при помощи которых данные переносили с одной ЭВМ на другую, а также запускали и устанавливали ОС Юникс на популярные тогда мини-ЭВМ PDP/11.

Рисунок 1. Мини-ЭВМ PDP/11

Расшифровка названия команды тоже относится к этим давно ушедшим временам. В языке IBM System/360 JCL был оператор DD 'Dataset Definition' (Определение набора данных), имя которого и получила вновь созданная команда.

Рисунок 2. Перфолента PDP/11

Расшифровок приходилось встречать много, в частности в рунете популярна расшифровка Disk Dump (не то разгрузка, не то загрузка диска) — версия слабая, так как в описываемые времена ни дисков, ни дампов (что бы под ними не понимали) еще не было. Гораздо ближе к сути команды шутливые расшифровки: «data destroyer» или «delete data», что можно перевести как «Доконай Диск» или «Добей Данные», потому что при неправильном использовании команды раздел или выходной файл мгновенно превращаются в хлам. Поскольку dd является инструментом для создания головных меток диска, загрузочных записей и тому подобных системных областей диска, наверняка многие жесткие диски и файловые системы были уничтожены неправильным применением dd.

Синтаксис, не похожий на команду Юникс

Из глубины времен пришел и синтаксис команды dd, не похожий ни на одну команду Юникс. Синтаксис команды dd кардинально отличается от синтаксиса большинства остальных команд Юникс. Благодаря своей уникальности он устоял против недавних попыток унифицировать синтаксис для всех программ командной строки. Так dd использует формат 'опция=значение', в то время как большинство команд Юникс используют формат ' -опция значение'. Кроме того, ввод команды dd определяется опцией «if» (input file), в то время как большинство команд просто используют само имя, без всяких опций. Существует мнение, что такой синтаксис основан на языке программирования IBM System/360 JCL, также ходят слухи, что этот синтаксис был создан как шутка; тем не менее, не было ни одной попытки написать замену программе dd, которая более походила бы на команду Юникс.

Даже говоря о формате синтаксиса dd 'опция=значение', я допускаю неточность. Опций в прямом смысле у этой команды всего две: –help и –version, их применение очевидно. Все остальные элементы синтаксиса называются операндами. Их не так уж и много, но многие столь сложно зависят друг от друга, что, если начать приводить их по списку, то получится очередной ман, разобраться в котором не просто. (Мне встречались даже маны с внутренними ссылками, но сильно это не помогало).

Поэтому в этой статье я делаю упор на примеры.

Для знакомства с большинством обычных действий с командой dd нужно начать с нескольких простых операндов.

Обычные варианты применения команды dd

Начнем знакомство с операндами команды dd:

if=filename (input file) Этот операнд задает входной файл; если он не указан, то по умолчанию используется стандартный ввод. Этим файлом может быть также файл (нода) устройства, например /dev/hda1, или специфические файлы типа /dev/zero.

of=filename (output file) Задает выходной файл; если он не указан, то по умолчанию используется стандартный вывод (экран монитора).

Знакомство с этими двумя операндами уже дает нам возможность использовать программу dd для копирования файлов.

Пример 1. Простое копирование.

```
# dd if=/home/ya/Desktop/shema.txt of=/home/ya/Desktop/shema.html
3+1 записей считано
3+1 записей написано
скопировано 1549 байт (1,5 kB), 0,427457 секунд, 3,6 kB/s
```

Как мы видим, программа dd сняла копию файла shema.txt и записала данные в файл shema.html. При этом она выдала нам сообщение.

Разберемся, о чем говорится в этом сообщении. Команда dd считывает и записывает блоками по N байт в каждом. Поскольку мы не задали в командной строке размер блока, то программа использовала размер блока по умолчанию, равный 512 байт. Всего скопировано 1549 байт — записано в последней строке сообщения. «3+1 записей считано» означает число полных блоков — 3 (3*512=1536) плюс один неполный блок, содержащий оставшиеся байты 1549-1536=13. То же относится и к строке «3+1 записей написано».

Теперь самое время разобраться с размером блока.

Сразу скажу, что блок команды dd не имеет никакого отношения к блокам данных файловой системы. Программа dd работает с необработанными «сырыми» (raw data) данными на низком уровне, т.е. на уровне секторов жесткого диска. А любая файловая система является надстройкой над этим уровнем. Правильнее было бы назвать блок буфером. Мы имеем два буфера — входной буфер и выходной буфер. Величина буфера задается операндом **bs** (block size) и исчисляется в байтах. Например, запись bs=512 означает, что нами установлен размер блока (буфера) в 512 байт. Кстати это значение и используется по умолчанию. Если величина блока не указана, dd использует блоки по 512 байт, что подходит для абсолютного большинства задач. Итак, величина буфера задана, dd создает 512-байтный буфер считывания, посылает единственный запрос на чтение со входного файла, затем создает 512-байтный буфер записи и посылает единственный запрос на запись в выходной файл. Давайте сравним работу программы dd и программы cp при копировании дискеты.

```
#  dd if=/dev/fd0 of=floppy.img bs=1474560
1+0 записей считано
1+0 записей написано
```

```
скопировано 1474560 байт (1,5 MB), 0,0438156 секунд, 33,7 MB/s
```

Может возникнуть вопрос: «А почему именно 1474560 байт, а не 1440 000, как принято считать размер дискеты?» Это значение складывается из: 2 головки, 80 цилиндров, 18 секторов/дорожку при длине сектора 512 байт. Итого: 2*80*18*512=1474560 байт.

Команде же

```
# cp /dev/fd0 floppy.img
```

требуется послать 360 запросов по 4096 байт (размер блока данных), что даст те же 360*4096=147560 байт. Для файла размером 1.44Мб это может показаться незначительным, но когда производятся операции с большими объемами данных, уменьшение числа системных вызовов приводит к существенному повышению быстродействия системы.

Итак, размер блока задается операндом

bs=n (байт)

Существуют и другие способы задания этого параметра, например при помощи знака <;x>;, который понимается как умножение. Только что мы вычисляли точный размер дискеты, тем же способом можно задать величину блока:

bs=2x80x18x512

или используя суффиксы:

bs=2x80x18b

где суффикс b означает 512, а вся запись аналогична предыдущей. Применяются следующие суффиксы:

```
w  означает 2 (в некоторых реализациях 4)
b  означает 512
k  означает 1024
```

Короче говоря, как часто бывает в Юниксе, все максимально запутано и требует недюжинной памяти. Но и это еще не все. Кроме bs, существуют еще три операнда размера блока:

ibs=n (байт) Этот операнд определяет размер входного блока. Правила его задания такие же, как для bs.

obs=n (байт) Этот операнд определяет размер выходного блока. Правила его задания такие же, как для bs.

bs является как бы старшим операндом, и если он задан, то **ibs** и **obs** игнорируются. ibs и obs нужны в тех случаях, когда необходимо задать различные значения входного и выходного блоков.

И четвертый операнд, связанный с размером блока:

cbs=n (байт) Определяет размер буфера (блока) преобразования. О нем мы будем говорить в главе, посвященной преобразованиям форматов данных.

Теперь, когда мы умеем задавать размеры блока, мы можем выполнить очень важную задачу — снять резервную копию Главной загрузочной записи — MBR. Все, у кого на компьютере имеется несколько операционных систем, должны иметь такую копию, и обновлять ее при каждом изменении системы. MBR находится в 0 секторе первого раздела жесткого диска и занимает вместе с таблицей разделов ровно 512 байт.

Для того чтобы скопировать MBR, следует познакомиться с еще одним операндом:

count=n (блоков) Этот операнд определяет количество блоков, подлежащих копированию. Например запись

bs=512 count=100

означает приказ скопировать 100 блоков по 512 байт каждый. Здесь также возможно применение суффиксов, например

bs=512 count=1k

означает, что будет скопировано 1024 блока по 512 байт каждый.

Внимание! Операнд count имеет дело с блоками, а не с байтами!

Команда, которая скопирует нам MBR в файл backup.mbr выглядит так:

```
# dd if=/dev/hda of=backup.mbr bs=512 count=1
1+0 записей считано
1+0 записей написано
скопировано 512 байт (512 B), 0,000358146 секунд, 1,4 MB/s
```

Резервная копия создана, теперь главное не забыть, где она лежит. И если какой-нибудь вредный дистрибутив криво установит нам загрузчик Grub, то мы спокойно проделаем обратную операцию:

```
# dd if=backup.mbr of=/dev/hda bs=512 count=1
```

И машина снова станет загружаться. Просмотреть полученную копию можно в любом двоичном (шестнадцатеричном) редакторе, например Khexedit или редакторе из Midnight Commander.

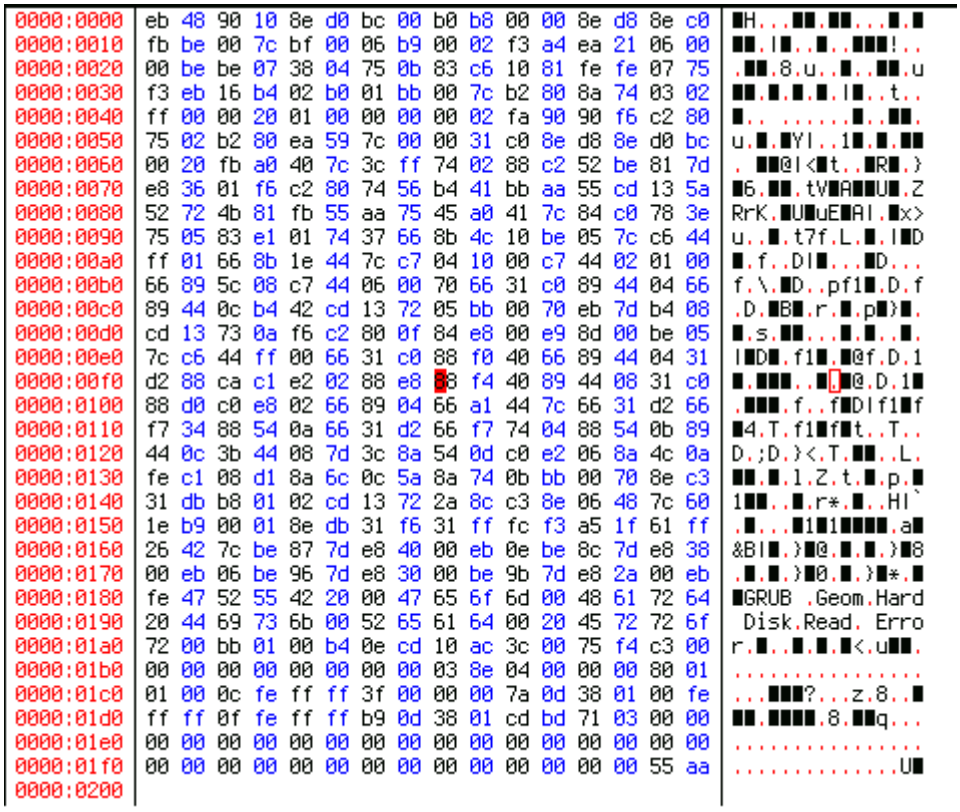

Рисунок 3. MBR полностью, все 512 байт

А если нам нужно не весь MBR, а только загрузочный код, который занимает первые 446 байт 0 сектора? Тогда придется изменить размер блока:

```
# dd if=/dev/hda of=boot-code.mbr bs=446 count=1
1+0 записей считано
1+0 записей написано
скопировано 446 байт (446 B), 0,0345156 секунд, 1,4 MB/s
```

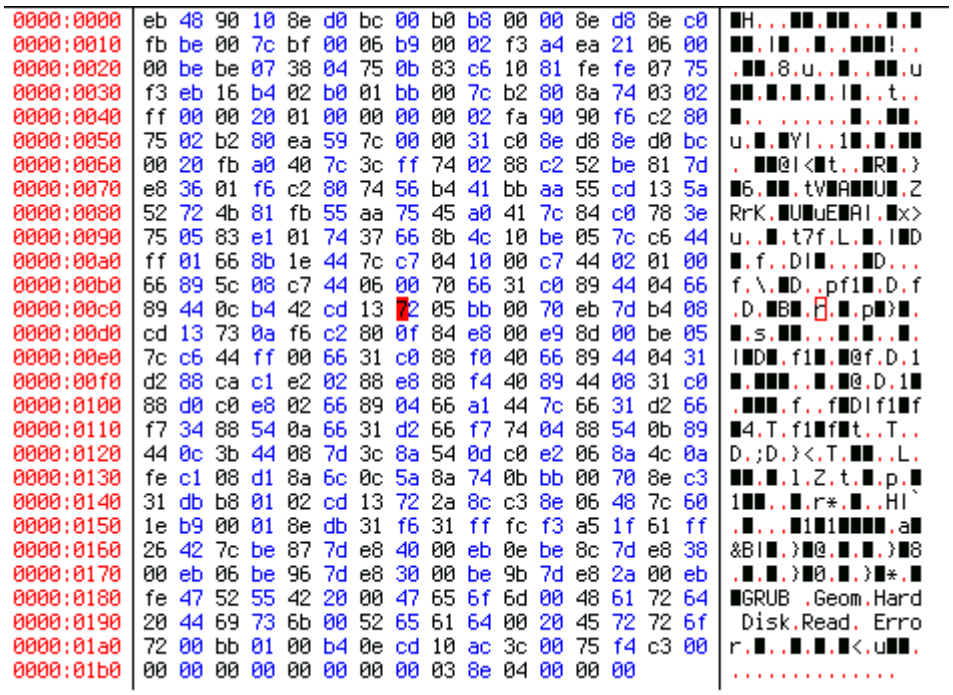

Рисунок 4. Загрузочная программа, занимающая первые 446 байт MBR

Если же мы хотим сохранить только таблицу разделов, а загрузочный код (скажем от Windows) нам больше не нужен? Для выполнения этой задачи нам нужно познакомиться еще с одним операндом:

skip=n (блоков) Этот операнд пропускает n(блоков) от начала входного (if) файла, а затем копирует указанное количество блоков.

Внимание! Операнд skip (как и count) имеет дело с блоками, а не с байтами! Поэтому размер блока следует выбирать вдумчиво. Что мы и сделаем:

```
# dd if=/dev/hda of=part-table.mbr bs=1 count=66 skip=446
66+0 записей считано
66+0 записей написано
скопировано 66 байт (66 B), 0,034515 секунд, 11,9 kB/s
```

Мы выбрали размер блока в 1 байт, и бесхитростно указываем нужные параметры в байтах.

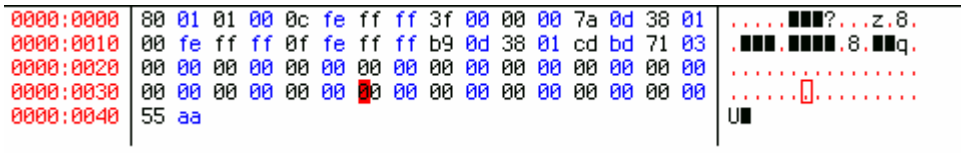

Рисунок 5. Таблица разделов вместе с сигнатурой (55 aa)

Может возникнуть вопрос: «Почему копировали 66 байт, хотя известно, что таблица разделов занимает 16*4=64 байта?» Дело в том, что последние два байта в таблице разделов занимает так называемая сигнатура (подпись) [55 aa], без которой ни один компьютер не опознает таблицу разделов. И поскольку мы делаем резервную копию, то, во избежание забывчивости и недоразумений при будущем восстановлении таблицы разделов, присоединили сигнатуру к таблице разделов.

Как вам нравится рисунок 5? Какой-то он непривычный (смещение на два байта) и неинформативный. Лучше бы иметь стандартную распечатку MBR, в котором первые 446 байт заменены, скажем, нулями. Если придется править вручную, будет намного удобнее. Команда dd позволяет и это. Только нужно применить еще один операнд:

seek=n (блоков) Этот операнд пропускает в выходном файле (of) n блоков, прежде чем начать туда запись.

Внимание! Операнд seek (как и skip и count) имеет дело с блоками, а не с байтами!

Место пропущенных блоков он заполняет нулями. Значит нам нужно применить такую команду:

```
# dd if=/dev/hda of=part-table.mbr bs=1 count=66 skip=446 seek=446
66+0 записей считано
66+0 записей написано
скопировано 66 байт (66 B), 0,034515 секунд, 12,9 kB/s
```

```
0000:0000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0040  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:00a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:00b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:00c0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ......▯.........
0000:00d0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:00e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:00f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0120  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0130  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0140  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0150  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0160  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0180  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:0190  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:01a0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:01b0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 80 01  ................
0000:01c0  01 00 0c fe ff ff 3f 00 00 00 7a 0d 38 01 00 fe  ...▮▮▮?...z.8..▮
0000:01d0  ff ff 0f fe ff ff b9 0d 38 01 cd bd 71 03 00 00  ▮▮.▮▮▮▮.8.▮▮q...
0000:01e0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0000:01f0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 aa  ..............U▮
0000:0200
```

Рисунок 6. Теперь таблица разделов приобрела привычный, удобопонятный вид. И сигнатура на месте.

Давайте на примерах рассмотрим многообразие применения команды dd, не слишком углубляясь в дебри синтаксиса.

Пример 2. Создание загрузочной дискеты из файла-образа.

Допустим, мы скачали из Интернета файл-образ чрезвычайно полезного загрузочного менеджера SBM (SmartBootManager) sbm.img. Теперь нужно создать загрузочную дискету SBM:

```
# dd if=/путь/sbm.img of=/dev/fd0
```

Никаких дополнительных операндов не требуется, поскольку файл-образ точно рассчитывается под величину дискеты.

Пример 3. Разрезать 10 мегабайтный файл file.10m на два пяти-мегабайтных:

```
dd if=file.10m of=file1.5m bs=1M count=5
dd if=file.10m of=file2.5m bs=1M skip=5


</file>
```

==== Пример 4. вывести на экран первые 100 байт содержимого файла schema.txt: ====

<code># dd if=Desktop/shema.txt bs=100 count=1

<;html>; <;head>; <;meta content=«text/html; charset=koi8-r» http-equiv=«Content-Type»>; <;title>; 1+0 записей считано 1+0 записей написано скопировано 100 байт (100 B), 0,00155746 секунд, 64,2 kB/s

Обратите внимание, что мы не указывали выходной файл (of). Поэтому команда dd по умолчанию воспользовалась стандартным выводом (экраном монитора).

Пример 5. Создание образа оптического диска:

```
dd if=/dev/cdrom of=image.iso bs=2k
```

При желании, с помощью полученного образа image.iso можно будет «прожечь» полноценный загрузочный диск (если, конечно, оригинал был загрузочным). Если копировать командой cp, то загрузочного диска не получишь.

Пример 6. Ускорение работы некоторых Live CD.

На «слабых» машинах современные Live CD работают медленно, иногда очень медленно. Если на жестком диске выделить раздел, скажем hda7 размером в 900Мб (для CD) и скопировать в этот раздел весь CD целиком, то во многих случаях, при загрузки с CD загрузчик примет наш раздел жесткого диска за сам CD и начнет загрузку с него. В этом случае скорость работы сильно повышается, ведь работа с жесткого диска намного быстрее, чем с любого CD привода.

```
# dd if=/dev/cdrom of=/dev/hda bs=2k
```

Этот фокус не всегда срабатывает, так как некоторые загрузчика не перебирают все файлы устройств по порядку. Например, Gentoo Live CD и Mandriva Spring Live CD, я только таким способом и смог запустить в графическом режиме на старом ноутбуке. А Mops Live CD на обман не поддается, и ищет чего-то другого.

Пример 7. Увеличить размер существующего файла до 1Гб без перезаписи:

```
dd if=/dev/zero of=file.name bs=1 count=0 seek=1G


</file>

==== Пример 8. Создание загрузочной дискеты Grub ====

Для создания загрузочной дискеты Grub, нужно записать файл stage1 в нулевой сектор дискеты, а файл stage2 в первый и последующие сектора. Файлы stage1 и stage2 обычно находятся в директории /boot/grub, если Grub у вас установлен, или в пакете,
```

если Grub еще не установлен. Я не стану указывать путь к этим файлам, он у всех разный.

<code># dd if=stage1 of=/dev/fd0 bs=512 count=1

Файл stage1 (загрузочный) занял свое место в загрузочном (0) секторе дискеты.

```
# dd if=stage2 of=/dev/fd0 bs=512 seek=1
153+1 records in
153+1 records out
```

Команда dd пропустила один сектор (поскольку там уже записаны данные stage1, то нулями она его не заполнила) и записала весь файл stage2. Теперь с этой дискеты можно загружать Grub, и с его помощью запускать любую ОС на вашей машине.

Пример 9. Создание загрузочного образа

Сходную задачу выполняет Makefile ядра Линукс при создании загрузочного образа (boot image). В Alpha Makefile /usr/src/linux/arch/alpha/boot/Makefile программа srmboot на выходе выдает команду:

```
# dd if=bootimage of=$(BOOTDEV) bs=512 seek=1 skip=1
```

Эта команда означает: пропустить первые 512 байт во входном файле (bootimage) (seek=1); начать запись со второго сектора устройства $(BOOTDEV) (seek=1). Типичное использование команды dd в том, чтобы пропустить исполняемую программу в заголовке, и начать запись в середине устройства, пропуская данные о томе и разделе. Нужно сказать, что малейшая ошибка в такого рода записях, может стоить вашему диску потери системных данных, поэтому будьте осторожны и проверьте все трижды.

Пример 10. Вывести на экран уже знакомый нам MBR.

Поскольку это двоичный файл, придется воспользоваться следующим конвейером:

```
# dd if=/dev/hda bs=512 count=1 2>;/dev/null | hexdump
```

0000000 48eb 1090 d08e 00bc b8b0 0000 d88e c08e 0000010 befb 7c00 00bf b906 0200 a4f3 21ea 0006 0000020 be00 07be 0438 0b75 c683 8110 fefe 7507 0000030 ebf3 b416 b002 bb01 7c00 80b2 748a 0203 0000040 00ff 2000 0001 0000 0200 90fa f690 80c2 0000050 0275 80b2 59ea 007c 3100 8ec0 8ed8 bcd0 0000060 2000 a0fb 7c40 ff3c 0274 c288 be52 7d81 0000070 36e8 f601 80c2 5674 41b4 aabb cd55 5a13 0000080 7252 814b 55fb 75aa a045 7c41 c084 3e78 0000090 0575 e183 7401 6637 4c8b be10 7c05 44c6 00000a0 01ff 8b66 441e c77c 1004 c700 0244 0001 00000b0 8966 085c 44c7 0006 6670 c031 4489 6604 00000c0 4489 b40c cd42 7213 bb05 7000 7deb 08b4 00000d0 13cd 0a73 c2f6 0f80 e884 e900 008d 05be 00000e0 c67c ff44 6600 c031 f088 6640 4489 3104 00000f0 88d2 c1ca 02e2 e888 f488 8940 0844 c031 0000100 d088 e8c0 6602 0489 a166 7c44 3166 66d2 0000110 34f7 5488 660a d231 f766 0474 5488 890b 0000120 0c44 443b 7d08 8a3c 0d54 e2c0 8a06 0a4c 0000130 c1fe d108 6c8a 5a0c 748a bb0b 7000 c38e 0000140 db31 01b8 cd02 7213 8c2a 8ec3 4806 607c 0000150 b91e 0100 db8e f631 ff31 f3fc 1fa5 ff61 0000160 4226 be7c 7d87 40e8 eb00 be0e 7d8c 38e8 0000170 eb00 be06 7d96 30e8 be00 7d9b 2ae8 eb00 0000180 47fe 5552 2042

4700 6f65 006d 6148 6472 0000190 4420 7369 006b 6552 6461 2000 7245 6f72 00001a0 0072 01bb b400 cd0e ac10 003c f475 00c3 00001b0 0000 0000 0000 0000 8e03 0004 0000 0180 00001c0 0001 fe0c ffff 003f 0000 0d7a 0138 fe00 00001d0 ffff fe0f ffff 0db9 0138 bdcd 0371 0000 00001e0 0000 0000 0000 0000 0000 0000 0000 0000 0000 00001f0 0000 0000 0000 0000 0000 0000 0000 aa55 0000200

Обратите внимание на сигнатуру. Она представлена байтами aa55. А на рисунке 3 мы видели в Khexedit'e 55aa. Если внимательно сравнить полученную распечатку с рисунком 3, то становится ясно, что байты в каждой паре поменялись местами: eb48 стала 48eb, 9010 стала 1090 и так далее. Дело в том, что существуют минимум две системы записи двоичных файлов: little endian и big endian. Khexedit, с которого был снят рисунок 3, работает в little endian. А какое-то звено в примененном нами конвейере, похоже работает в big endian (Кстати, определить какую систему записи байтов использует ваша машина, можно с помощью небольшой программы. Смотрите приложение [1]). Чтобы получить распечатку в уже привычном нам виде (как на рис. 3) нам необходимо использовать новый операнд:

conv=значение Этот операнд совершает преобразования формата копируемых файлов. Он работает не с числами, а со значениями (или иначе, фильтрами), список которых весьма обширен. Мы рассмотрим этот список в следующей главе, а пока применим одно из них:

conv=swab Это преобразование переставляет байты в каждой входной паре.

```
# dd if=/dev/hda bs=512 count=1 conv=swab 2>;/dev/null | hexdump
```

0000000 eb48 9010 8ed0 bc00 b0b8 0000 8ed8 8ec0 0000010 fbbe 007c bf00 06b9 0002 f3a4 ea21 0600 0000020 00be be07 3804 750b 83c6 1081 fefe 0775 0000030 f3eb 16b4 02b0 01bb 007c b280 8a74 0302 0000040 ff00 0020 0100 0000 0002 fa90 90f6 c280 0000050 7502 b280 ea59 7c00 0031 c08e d88e d0bc 0000060 0020 fba0 407c 3cff 7402 88c2 52be 817d 0000070 e836 01f6 c280 7456 b441 bbaa 55cd 135a 0000080 5272 4b81 fb55 aa75 45a0 417c 84c0 783e 0000090 7505 83e1 0174 3766 8b4c 10be 057c c644 00000a0 ff01 668b 1e44 7cc7 0410 00c7 4402 0100 00000b0 6689 5c08 c744 0600 7066 31c0 8944 0466 00000c0 8944 0cb4 42cd 1372 05bb 0070 eb7d b408 00000d0 cd13 730a f6c2 800f 84e8 00e9 8d00 be05 00000e0 7cc6 44ff 0066 31c0 88f0 4066 8944 0431 00000f0 d288 cac1 e202 88e8 88f4 4089 4408 31c0 0000100 88d0 c0e8 0266 8904 66a1 447c 6631 d266 0000110 f734 8854 0a66 31d2 66f7 7404 8854 0b89 0000120 440c 3b44 087d 3c8a 540d c0e2 068a 4c0a 0000130 fec1 08d1 8a6c 0c5a 8a74 0bbb 0070 8ec3 0000140 31db b801 02cd 1372 2a8c c38e 0648 7c60 0000150 1eb9 0001 8edb 31f6 31ff fcf3 a51f 61ff 0000160 2642 7cbe 877d e840 00eb 0ebe 8c7d e838 0000170 00eb 06be 967d e830 00be 9b7d e82a 00eb 0000180 fe47 5255 4220 0047 656f 6d00 4861 7264 0000190 2044 6973 6b00 5265 6164 0020 4572 726f 00001a0 7200 bb01 00b4 0ecd 10ac 3c00 75f4 c300 00001b0 0000 0000 0000 0000 038e 0400 0000 8001 00001c0 0100 0cfe ffff 3f00 0000 7a0d 3801 00fe 00001d0 ffff 0ffe ffff b90d 3801 cdbd 7103 0000 00001e0 0000 0000 0000 0000 0000 0000 0000 0000 0000 00001f0 0000 0000 0000 0000 0000 0000 0000 55aa 0000200

Теперь MBR принял привычный для нас вид, и мы можем заняться рассмотрением других преобразований, осуществляемых командой dd.

Преобразования или конвертации

В первых строках любого мана dd говорится: «Команда dd копирует указанный входной файл в

указанный выходной, совершая при необходимости преобразования.» Причем совершаются эти преобразования одновременно с копированием, как говорят по-английски «на лету».

Одной из первоначальных задач команды dd была работа с 1/2« ленточными накопителями, которые применялись для переноса данных с одной машины на другую, а также для запуска и установки ОС Юникс на машинах PDP/11. Те времена прошли, а 9-дорожечный формат остался. И остался до тонкостей разработанный синтаксис этих преобразований.

Как мы уже знаем, ведает конвертациями операнд conv.

conv=значение В некоторых манах значение называют фильтром. Список этих значений-фильтров сильно варьирует от системы к системе. Тут вообще уместно сделать небольшое отступление для тех, кто захочет досконально разобраться с командой dd.

Существует несколько реализаций программы dd. В Линукс системах чаще всего встречается GNU Coreutils dd. В BSD своя реализация, в Solaris своя. Я имею сильное подозрение, что все они похожи как близнецы братья, только маны разные. Самые толковые, подробные и запутанные маны от Solaris, за ними идут BSD, а самые коротенькие от GNU. Похоже, что они не описывают и половины способностей программы. Я приведу здесь выдержки из нескольких манов, а в приложении [2] дам перевод самого длинного.

Итак значения или фильтры для операнда **conv=значение**:

Внимание! Значения ascii, ebcdic и ibm — взаимоисключающие.

- **ascii** Преобразует (конвертирует) кодировку EBCDIC (Extended Binary Coded Decimal Interchange Code) в ASCII.
- **ebcdic** Преобразует кодировку ASCII в EBCDIC. При преобразовании записей ASCII фиксированной длины, не разделенных символами новой строки, необходимо организовать конвейер и предварительно обрабатывать данные командой dd conv=unblock.
- **ibm** Немного другое преобразование кодировки ASCII в EBCDIC.

Сейчас нам придется прервать список значений операнда conv, чтобы вспомнить о четвертом операнде размера блока. Когда мы обсуждали эти операнды (bs, ibs, obs и cbs), то ничего не сказали про операнд:

cbs=n (байт) Определяет размер буфера (блока) преобразования. Задает размер блока преобразования (n байтов, по умолчанию — 0) для операндов block и unblock. Если операнд cbs= не указан или задает значение 0, использование операндов block и unblock дает неопределенные результаты.

Познакомившись с операндом cbs, мы можем продолжать список значений-фильтров операнда conv:

Внимание! Значения block и unblock — взаимоисключающие.

- **block** Рассматривает входной файл как последовательность записей переменной длины, завершаемых символами новой строки или символом конца файла (EOF), независимо от размера входного блока. Каждая запись преобразуется в запись фиксированной длины, задаваемой размером блока преобразования. Все символы новой строки из входной строки удаляются; к строкам при необходимости добавляются пробелы, чтобы заполнить блок. Строки, длина которых превышает размер блока преобразования, усекаются до

этого размера; утилита сообщает о количестве усеченных строк.
- **unblock** Преобразует записи фиксированной длины в записи переменной длины. Читает количество байтов, равное размеру блока преобразования (или оставшиеся байты входного файла, если их меньше, чем байтов в блоке преобразования), удаляет все хвостовые пробелы и добавляет символ новой строки.

Чтобы показать, зачем нужны все эти блоки-анблоки, приведу конкретный пример. Повторюсь, что поначалу задача команды dd была работа с 1/2» ленточными накопителями, которые применялись для переноса данных с одной машины на другую. Те времена прошли, а 9-дорожечный формат остался. И команда dd не имеет себе равных в доступе к освященным веками 1/2« девятидорожечным ленточным накопителям. На современных SCSI ленточных устройствах разделение на блоки и распаковка блоков больше не обязательны, так как устройства читают и записывают блоками данных по 512 байт.

Тем не менее, 9-дорожечный 1/2» ленточный формат позволяет создавать блоки различной длины, поэтому его невозможно прочесть при помощи команды cp. Команда dd позволяет настраивать размер входных и выходных блоков, и даже может считывать блоки варьируемой длины, если установить размер входного буфера заведомо больший, чем самый большой блок на ленте. Короткие блоки считываются, и dd весело и без жалоб копирует их в выходной файл, просто подсчитывая количество полных и неполных блоков.

Есть еще EBCDIC наборы данных, перенесенные с таких операционных систем как MVS, почти всегда они представлены образами 80-колонных перфокарт Холлерита! Команда dd без проблем конвертирует их в ASCII.

Пример 11. Конвертация EBCDIC

Конвертация EBCDIC 80-значных фиксированной длины записей в записи ASCII переменной длины с подставлением символов новых строк:

```
# dd bs=10240 cbs=80 conv=ascii,unblock if=/dev/st0 of=ascii.out
40+0 records in
38+1 records out
```

Фиксированная длина записи указана как параметр cbs=80; входные и выходные блоки установлены параметром bs=10240. Конвертация EBCDIC-в-ASCII и фиксированная-в-переменную длины записей включены параметрами conv=ascii,unblock.

Получив некоторое представление о значениях block и unblock, давайте вернемся к продолжению списка значений-фильтров операнда conv:

Внимание! Значения lcase и ucase — взаимоисключающие.

- **lcase** Преобразует все символы верхнего регистра, в соответствии со значением LC_CTYPE, в нижний регистр. Символы, для которых преобразование в локали не задано, не изменяются.
- **ucase** Сами понимаете, наоборот

Пример 12. Преобразование верхнего регистра в нижний

Создайте текстовой файл uppercase.txt следующего содержания: «THESE LETTERS ARE UPPERCASE» (что по-нашему значит: «Все эти буквы в верхнем регистре»). Затем наберите команду:

```
$ dd if=uppercase.txt conv=lcase
these letters are uppercase
```

Поскольку мы не указали выходной файл (of), то получили преобразование на стандартный выход. Интересно, что с русскими буквами конвертация не осуществляется, видимо что-то нужно изменить в переменной окружения LC_CTYPE.

Вернемся к списку значений операнда conv:

- **swab** Уже знакомое нам значение. Переставляет байты в каждой входной паре. Если в текущей входной записи нечетное количество байтов, последний байт игнорируется.
- **noerror** Не останавливать обработку в случае ошибки чтения. При возникновении ошибки чтения в стандартный поток ошибок выдается диагностическое сообщение, а затем — текущий входной и выходной блок в том же формате, что и при завершении работы. Если задано преобразование sync, недостающие входные данные будут заменены нулевыми байтами; в противном случае, сбойный входной блок просто не попадает в выходной файл.

Это чрезвычайно важное значение. Ведь делая резервные копии больших файлов, скажем жестких дисков, вся работа пошла бы насмарку из-за крохотной ошибки в одном байте.

- **sync** Дополнять каждый входной блок до размера буфера, задаваемого операндом ibs=, добавляя нулевые байты. (Если указан также операнд block или unblock, добавляются пробелы, а не нулевые байты.)

Сделаем копию жесткого диска, используя два последних значения операнда conv,

Пример 13. Создание резервной копии жесткого диска прямо на другой диск:

```
# dd if=/dev/hda of=/dev/sda conv=noerror,syncbs=4k
```

Выбор размера блока не случаен: при маленьких блоках процесс копирования затянется, Можно задать большие блоки, но тогда в случае ошибок, оставшиеся после ошибки части блока будут заполнены нулями. Поэтому чем больше блоки, тем больше информации вы потеряете.

Уж, коль скоро зашла речь о резервном копировании, вот еще пример, не относящийся к преобразованиям, но пусть они будут рядом.

Пример 14. Создание резервного образа жесткого диска:

```
# dd if=/dev/hda | gzip>; /mnt/hdb1/system_drive_backup.img.gz
```

Команда dd создает образ первого жесткого диска и по конвейеру (не забудьте про стандартный вывод по умолчанию) передает программе сжатия gzip. Сжатый образ затем помещается в файл system_drive_backup.img.gz, находящийся на другом диске (hdb1). Чтобы произвести обратное действие:

```
# gzip -dc /mnt/hdb1/system_drive_backup.img.gz | dd of=/dev/hda
```

Программа gzip распаковывает (опция -d) файл, передает результат на стандартный выход (опция -c), по конвейеру данные поступают программе dd, которая записывает выходной файл (устройство /dev/hda).

Незаметно мы рассмотрели почти все значения (фильтры) операнда conv. Тут и ману конец. Но сначала несколько правил относящихся к взаимодействию операндов.

Правило 1. Все операнды обрабатываются до начала чтения входных данных. В практическом смысле это значит, что можно писать операнды в любом порядке — результат будет один. Скажем команда:

```
# dd if=/dev/hda of=/dev/sda conv=noerror,sync bs=4k
```

равносильна команде

```
# dd bs=4k conv=noerror,sync if=/dev/hda of=/dev/sda
```

Правило 2. Если операнды (кроме conv=) указаны более одного раза, будет использоваться значение из последней пары операнд=значение.

Еще примеры возможностей команды dd

Пример 15. Уничтожение всех данных в разделе hdc2, размером 1Гб:

Чтобы было невозможно восстановить содержимое этого раздела, заполним его нулями от первого до последнего байта.

```
# dd if=/dev/zero of=/dev/hdc2 bs=1M
dd: запись `/dev/hdc2': No space left on device
957+0 записей считано
956+0 записей написано
скопировано 1003484160 байт (1,0 GB), 44,7713 секунд, 22,4 MB/s
```

А можно псевдослучайными байтами:

```
# dd if=/dev/urandom of=/dev/hdc2 bs=1M
dd: запись `/dev/hdc2': No space left on device
```

```
957+0 записей считано
956+0 записей написано
скопировано 1003484160 байт (1,0 GB), 1050,83 секунд, 955 kB/s
```

Давайте проверим, что получилось. Возьмем на вскидку 1001 сектор раздела hdc2, который мы только что заполнили псевдослучайными байтами:

```
# dd if=/dev/hdc2 bs=512 count=1 skip=1000

???:?+K▒^??8???/-O?L*lJ?N`????????_
?7$?.pXi(\kN]c??6@lJ2IM]Z?9iAVlm?6?wCm?;s???r?2\?&Sd>;???!n??!????P???9!s-{?
 0??-uK???H?a>;?F[?S[???▒w?JP(wt~W1????D? }E?cF>;|?8{9 Q?t%7AnFtu
j?J??L? xv?'??_bs??TP   }Du?
1+0 записей считано
1+0 записей написано
скопировано 512 байт (512 B), 0,0114453 секунд, 44,7
```

Действительно, ахинея! Справедливости ради нужно заметить, что заполнение нулями при прочих равных условиях занимает на порядок меньше времени.

Таким же способом можно прочесть любой сектор, или группу секторов любого носителя.

Особые случаи применения команды dd

Команда dd применяется в судебной и криминалистической практике. Самое очевидное применение — снятие точных (побайтовых) копий с жестких дисков подозреваемого. Работать с ними удобнее (не нужно подключать диски физически), да и места займет немного.

```
# dd if=/dev/hda of=/dev/case10img1
```

Другой случай. В руках криминалистов оказалась магнитная лента, подлежащая исследованию. Но неизвестен размер блока на этой пленке, а знание правильного размера многократно ускорит процесс прочтения. Можно попытаться определить размер блока при помощи команды dd:

```
# dd if=/dev/st0 ibs=128 of=/dev/case10img1 obs=1 count=1
```

задав несуразный размер входного блока (ibs=128), мы провоцируем программу выдать сообщение об ошибке, в котором будет указан истинный размер блока.

Еще одно применение команды dd состоит в возможности порезать большой объем данных (жесткий диск) на кусочки, удобные для записи на другие носители (например DVD):

```
# dd if=/dev/st0 bs=1M count=4000 of=/dev/case10img1
```

dd if=/dev/st0 bs=1M count=4000 skip=4000 of=/dev/case10img2

dd if=/dev/st0 bs=1M count=4000 skip=8000 of=/dev/case10img3

dd if=/dev/st0 bs=1M count=4000 skip=12000 of=/dev/case10img4

И так далее...

Команда dd и MS Windows

Существует реализация команды dd для Win32. Читайте статью Команда dd для Windows

К читателям

Уважаемые коллеги, если вы знаете, или прочитали про другие возможности применения команды dd, не сочтите за труд написать мне. Только пожалуйста, не ограничивайтесь одной командной строкой, а, по возможности, объясните что там к чему. Мой e-mail: yakwiat@yandex.ru.

Приложения

[1] Приложение 1. Как определить порядок записи байтов в вашей машине?

1. Создать текстовой файл testendian.c:

```
#include
unsigned short x = 1; /* 0x0001 */
int main(void)
{
  printf("%s\n", *((unsigned char *) &x) == 0 ? "big-endian" : "little-
endian");
  return 0;
}
```

2. Скомпилировать:

$ gcc -o testendian testendian.c

2. Запустить:

```
$ ./testendian
little-endian
```

Вывод: ваша машина использует little-endian порядок байтов.

[2] Приложение 2. Страницы ман команды dd для ОС Solaris (самые полные) в переводе на русский В. Кравчука.

НАЗВАНИЕ

dd — преобразование и копирование файла

СИНТАКСИС /usr/bin/dd [операнд=значение ...]

ОПИСАНИЕ

Команда dd копирует указанный входной файл в указанный выходной, совершая при необходимости преобразования. По умолчанию используются стандартный входной и стандартный выходной потоки. Можно задавать размеры блоков во входном и выходном файле, чтобы учесть особенности физического ввода-вывода. Размер блоков задается в байтах; значение размера может заканчиваться суффиксами k, b или w, задающими умножение на 1024, 512 и 2, соответственно. Числа можно также разделять символами x, интерпретируемыми как умножение.

Команда dd будет читать входной файл по одному блоку заданного размера; затем она обрабатывает данные, формируя выдаваемые блоки данных, которые могут быть меньше требуемого размера. Команда dd применяет любые заданные преобразования и записывает получившиеся данные в выходной файл блоками заданного для вывода размера.

Операнд cbs используется только если задано преобразование ascii, asciib, unblock, ebcdic, ebcdicb, ibm, ibmb или block. В первых двух случаях символы cbs копируются в буфер преобразования, выполняется любое указанное преобразование символов, хвостовые пробелы удаляются и перед посылкой строки в выходной файл добавляется символ новой строки. В последних трех случаях символы вплоть до символа новой строки считываются в буфер преобразования и дополняются пробелами для получения записи указанного размера. Предполагается, что файлы ASCII содержат символы новой строки. Если значение cbs не задано или равно 0, опции ascii, asciib, ebcdic, ebcdicb, ibm и ibmb преобразуют набор символов, не изменяя структуру блоков входного файла; опции unblock и block приводят к простому копированию файла.

После завершения работы dd возвращает количество полных или частичных прочитанных и выданных блоков.

ОПЕРАНДЫ

Поддерживаются следующие операнды:

of=файл Задает выходной файл; по умолчанию используется стандартный выходной поток. Если одновременно не задано преобразование seek=expr, выходной файл будет усекаться перед началом копирования, если только не указан операнд conv=notrunc. Если указан операнд seek=expr, но не указан conv=notrunc, в результате в выходном файле останутся только блоки, пропущенные командой dd. (Если размер пропускаемой части плюс размер входного файла меньше, чем прежний размер выходного файла, в результате копирования выходной файл станет меньше.)

ibs=n Задает размер входного блока — n байтов (по умолчанию — 512).

obs=n Задает размер выходного блока — n байтов (по умолчанию — 512).

bs=n Устанавливает размеры входного и выходного блока равными n байтов, переопределяя установки ibs= и obs=. Если не указаны никакие изменения, кроме sync, noerror и notrunc, каждый входной блок будет непосредственно копироваться на выход, а не собираться из меньших блоков.

cbs=n Задает размер блока преобразования (n байтов, по умолчанию — 0) для операндов block и unblock. Если операнд cbs= не указан или задает значение 0, использование операндов block и unblock дает неопределенные результаты.

Эта опция используется только если указано преобразование ASCII или EBCDIC. Для преобразований ascii и asciib входные данные обрабатываются так же, как и для операнда unblock, но символы преобразуются в ASCII перед удалением хвостовых пробелов. Для преобразований ebcdic, ebcdicb, ibm и ibmb входные данные обрабатываются так же, как и для операнда block, но символы преобразуются в EBCDIC или IBM EBCDIC после добавления хвостовых пробелов.

files=n Копирует и конкатенирует n входных файлов, прежде чем завершить работу (имеет смысл только если входные данные берутся с ленты или другого аналогичного устройства).

skip=n Пропускает n входных блоков (используя указанный размер блока) перед началом копирования. Если по файлу можно перемещаться, реализация dd прочитает или просто пропусти блоки; если же перемещаться по файлу нельзя, блоки будут прочитаны и проигнорированы.

iseek=n Перемещается на n блоков с начала входного файла перед копированием (походит для файлов на диске, где skip может работать весьма медленно).

oseek=n Перемещается на n блоков от начала выходного файла перед копированием.

seek=n Пропускает n блоков (используя заданный размер выходного блока) с начала выходного файла перед копированием. Если по файлу нельзя перемещаться, существующие блоки будут прочитаны, а пространство от текущего конца файла до указанного смещения (если смещение задано) будет заполнено нулевыми байтами; если же по файлу можно перемещаться, реализация dd пропустит данные вплоть до указанного смещения или прочитает блоки так, как описано выше.

count=n Копирует только n входных блоков.

conv=значение[,значение…] Здесь значения берутся из следующего списка:

ascii Преобразует кодировку EBCDIC в ASCII.

asciib Преобразует кодировку EBCDIC в ASCII, используя BSD-совместимые преобразования символов.

ebcdic Преобразует кодировку ASCII в EBCDIC. При преобразовании записей ASCII фиксированной длины, не разделенных символами новой строки, необходимо организовать конвейер и предварительно обрабатывать данные командой dd conv=unblock.

ebcdicb Преобразует кодировку ASCII в EBCDIC, используя BSD-совместимые преобразования символов. При преобразовании записей ASCII фиксированной длины, не разделенных

символами новой строки, необходимо организовать конвейер и предварительно обрабатывать данные командой dd conv=unblock.

ibm Немного другое преобразование кодировки ASCII в EBCDIC. При преобразовании записей ASCII фиксированной длины, не разделенных символами новой строки, необходимо организовать конвейер и предварительно обрабатывать данные командой dd conv=unblock.

ibmb Немного другое преобразование кодировки ASCII в EBCDIC, использующее BSD-совместимые преобразования символов. При преобразовании записей ASCII фиксированной длины, не разделенных символами новой строки, необходимо организовать конвейер и предварительно обрабатывать данные командой dd conv=unblock.

Значения ascii (или asciib), ebcdic (или ebcdicb) и ibm (или ibmb) — взаимоисключающие.

block Рассматривает входной файл как последовательность записей переменной длины, завершаемых символами новой строки или символом конца файла (EOF), независимо от размера входного блока. Каждая запись преобразуется в запись фиксированной длины, задаваемой размером блока преобразования. Все символы новой строки из входной строки удаляются; к строкам при необходимости добавляются пробелы, чтобы заполнить блок. Строки, длина которых превышает размер блока преобразования, усекаются до этого размера; утилита сообщает о количестве усеченных строк.

unblock Преобразует записи фиксированной длины в записи переменной длины. Читает количество байтов, равное размеру блока преобразования (или оставшиеся байты входного файла, если их меньше, чем байтов в блоке преобразования), удаляет все хвостовые пробелы и добавляет символ новой строки.

Значения block и unblock — взаимоисключающие.

lcase Преобразует все символы верхнего регистра, в соответствии со значением LC_CTYPE, в нижний регистр. Символы, для которых преобразование в локали не задано, не изменяются.

ucase

Значения lcase и ucase — взаимоисключающие.

swab Переставляет байты в каждой входной паре. Если в текущей входной записи нечетное количество байтов, последний байт игнорируется.

noerror Не останавливать обработку в случае ошибки чтения. При возникновении ошибки чтения в стандартный поток ошибок выдается диагностическое сообщение, а затем — текущий входной и выходной блок в том же формате, что и при завершении работы. Если задано преобразование sync, недостающие входные данные будут заменены нулевыми байтами; в противном случае, сбойный входной блок просто не попадает в выходной файл.

notrunc Не усекать выходной файл. Сохраняет блоки выходного файла, не перезаписываемые явно при этом вызове dd. (См. также предшествующий операнд of=файл.)

sync Дополнять каждый входной блок до размера буфера, задаваемого операндом ibs=, добавляя нулевые байты. (Если указан также операнд block или unblock, добавляются пробелы, а не нулевые байты.)

Если операнды (кроме conv=) указаны более одного раза, будет использоваться значение из

последней пары операнд=значение.

Для операндов bs=, cbs=, ibs= и obs= необходимо указать выражение, задающее размер в байтах. Это выражение может быть:

положительным целым десятичным числом положительным целым десятичным числом с суффиксом k, задающим умножение на 1024 положительным целым десятичным числом с суффиксом b, задающим умножение на 512 двумя или более положительными десятичными числами (с или без суффиксов k или b), разделенными символом x, задающим произведение соответствующих значений.

Все операнды обрабатываются до начала чтения входных данных.

ИСПОЛЬЗОВАНИЕ

Описание работы утилиты dd с файлами, размер которых превосходит 2 Гбайта (2**31 байтов) см. на странице справочного руководства largefile(5). ПРИМЕРЫ Пример 1: Копирование с одного стримера на другой: В следующем примере выполняется копирование со стримера 0 на стример 1 с использованием стандартных имен соответствующих устройств. example% dd if=/dev/rmt/0h of=/dev/rmt/1h Пример 2: Отсечение первых 10 байтов стандартного входного потока В следующем примере удаляются первые 10 байтов стандартного входного потока. example% dd ibs=10 skip=1 Пример 3: Чтение ленты в текстовый (ASCII) файл В следующем примере читается лента с данными в кодировке EBCDIC, сблокированными по десять 80-байтовых образов, в ASCII-файл x: example% dd if=/dev/tape of=x ibs=800 cbs=80 conv=ascii,lcase Пример 4: Использование conv=sync для записи на ленту В следующем примере используется синхронизация (conv=sync) при записи на ленту: example% tar cvf — . | compress | dd obs=1024k of=/dev/rmt/0 conv=sync ПЕРЕМЕННЫЕ СРЕДЫ Описание переменных среды LC_CTYPE, LC_MESSAGES и NLSPATH, влияющих на работу dd, см. на странице справочного руководства environ(5). СТАТУС ВЫХОДА Программа завершается со следующими значениями статуса выхода: 0 Входной файл успешно скопирован.
>;0 Произошла ошибка. Если при чтении произошла ошибка и не указано преобразование noerror, частично сформированный выходной блок будет записан в выходной файл, будет выдано диагностическое сообщение и копирование будет прекращено. При возникновении любой другой ошибки будет выдано диагностическое сообщение и копирование будет прекращено. ССЫЛКИ cp(1), sed(1), tr(1), attributes(5), environ(5), largefile(5) ДИАГНОСТИКА f+p records in(out) количество полностью и частично прочитанных (записанных) блоков ПРИМЕЧАНИЯ Не используйте dd для копирования файлов из одной файловой системы в другую, если в этих файловых системах разные размеры блоков. При использовании блочного устройства для копирования файлов последний блок будет дополнен нулевыми байтами до границы блока. Когда утилита dd читает из программного канала и заданы операнды ibs=X и obs=Y, результат будет всегда организован в блоки размера Y. Если используется операнд bs=Z, выходные блоки всегда имеют такой размер, как прочитанные из входного потока. При использовании dd для копирования файлов на ленту, размер файла должен быть кратным размеру сектора на устройстве (например, 512 байтов). Для копирования на ленту файлов произвольного размера используйте команды tar(1) или cpio(1). При получении сигнала SIGINT команда dd перед завершением работы выдаст информацию о состоянии в стандартный поток ошибок. Для остальных сигналов используются стандартные обработчики. <note>Взято отсюда: **http://linux-mania.narod.ru/human/dd.html</note> Наверх <note important> В**

моей WIKI постоянно ведётся какая-то работа со статьями.
Если у вас возникли вопросы или замечания,
можете их отправлять на почту support@mihanik.net** </note>

From:
https://wiki.mihanik.net/ - **wiki.mihanik.net**

Permanent link:
https://wiki.mihanik.net/doku.php/%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B:linux:dd_-_%D0%BA%D0%BE%D0%BC%D0%B0%D0%BD%D0%B4%D0%B0_%D0%BA%D0%BE%D1%82%D0%BE%D1%80%D0%B0%D1%8F_%D0%B1%D0%B5_%D0%BF%D0%BE%D1%85%D0%BE%D0%B6%D0%B0_%D0%BD%D0%B0_%D0%B4%D1%80%D1%83%D0%B3%D0%B8%D0%B5

Last update: **2022/06/20 09:24**