|  |
| --- |
| time dd if=/dev/zero of=test02 bs=4096 count=2441406 oflag=direct  **bs=block-size(Bytes) count=number-of-blocks**  if=FILE  if=/dev/zero (if=/dev/input.file) : The name of the input file you want dd the read from  of=FILE  of=/tmp/test1.img (of=/path/to/output.file) : The name of the output file you want dd write the input.file to  **conv= Each CONV symbol may be**:  sync pad every input block with NULs to ibs-size; when used with block or unblock, pad with spaces rather than  NULs  **fdatasync**  **physically write output file data before finishing**  fsync likewise, but also write metadata  **oflag= Each FLAG symbol may be**:  append append mode (makes sense only for output; conv=notrunc suggested)  **direct** use direct I/O for data  directory  fail unless a directory  **dsync** use synchronized I/O for data: Use synchronized I/O for data. Do not skip this option. This option get rid of caching and gives you good and accurate results  sync likewise, but also for metadata |

|  |
| --- |
| Sample outputs:  1000+0 records in 1000+0 records out 512000 bytes (512 kB) copied, 0.60362 s, 848 kB/s |

|  |
| --- |
| Test write performance real case:  dd if=/dev/zero of=test02 bs=4096 count=10 conv=fdatasync oflag=dsync  10+0 records in  10+0 records out  40960 bytes (41 kB) copied, 0.243703 s, 168 kB/s  Test read performance real case: |

|  |
| --- |
| **Ways in which you can invoke 'dd' to test the write speed:**   1. dd bs=1M count=256 if=/dev/zero of=test 2. dd bs=1M count=256 if=/dev/zero of=test; sync 3. dd bs=1M count=256 if=/dev/zero of=test conv=fdatasync 4. dd bs=1M count=256 if=/dev/zero of=test oflag=dsync   **What is the difference between those?**   1. dd bs=1M count=256 if=/dev/zero of=test  The default behaviour of dd is to not “sync” (i.e. not ask the OS to completely write the data to disk before dd exiting). The above command will just commit your 128 MB of data into a RAM buffer (write cache) – this will be really fast and it will show you the hugely inflated benchmark result right away. However, the server in the background is still busy, continuing to write out data from the RAM cache to disk. 2. dd bs=1M count=256 if=/dev/zero of=test; sync  Absolutely identical to the previous case, as anyone who understands how \*nix shell works should surely know that adding a ; sync does not affect the operation of previous command in any way, because it is executed independently, after the first command completes. So your (inflated) MB/sec value is already printed on screen while that sync is only preparing to be executed. 3. dd bs=1M count=256 if=/dev/zero of=test conv=fdatasync  This tells dd to require a complete “sync” once, right before it exits. So it commits the whole 256 MB of data, then tells the operating system: “OK, now ensure this is completely on disk”, only then measures the total time it took to do all that and calculates the benchmark result. 4. dd bs=1M count=256 if=/dev/zero of=test oflag=dsync  Here dd will ask for completely synchronous output to disk, i.e. ensure that its write requests don’t even return until the submitted data is on disk. In the above example, this will mean sync'ing once per megabyte, or 256 times in total. It will be the slowest mode, as the write cache is basically unused at all in this case.   **Which one do you recommend to use?**  This behaviour is perhaps the closest to the way real-world tasks behave:   * dd bs=1M count=256 if=/dev/zero of=test conv=fdatasync   If your server or VPS is really fast and the above test completes in a second or less, try increasing the count= number to 1024 or so, to get a more accurate averaged result. |