**Linux and Unix usefull commands**

1. **DD**

**About dd**

The **dd** command copies a file, converting the format of the data in the process, according to the operands specified.

On Unix, device drivers for hardware (such as hard disk drives) and special device files (such as /dev/zero and /dev/random) appear in the file system just like normal files; dd can also read and/or write from/to these files, provided that function is implemented in their respective driver. As a result, dd can be used for tasks such as backing up the boot sector of a hard drive, and obtaining a fixed amount of random data. The dd program can also perform conversions on the data as it is copied, including byte order swapping and conversion to and from the ASCII and EBCDIC text encodings.[2]

The name dd is an allusion to the DD statement found in IBM's Job Control Language (JCL),[3][4] in which the initials stand for "Data Definition".[5] The command's syntax resembles the JCL statement more than it does other Unix commands, so the syntax may have been a joke.[3]

Originally intended to convert between ASCII and EBCDIC, dd first appeared in Version 5 Unix.[6] The dd command is specified by IEEE Std 1003.1-2008, which is part of the Single UNIX Specification.

**dd syntax**

dd [*OPERAND*]...

dd *OPTION*

Man pages for more information on options.

**Options**

|  |  |
| --- | --- |
| **--help** | Display help and exit. |
| **--version** | Display version information and exit. |

**Numerical Suffixes**

*BLOCKS* and *BYTES* may be followed by the following multiplicative suffixes:

**c**=1  
**w**=2  
**b**=512  
**kB**=1000  
**K**=1024  
**MB**=1000\*1000  
**M**=1024\*1024  
**xM**=M  
**GB**=1000\*1000\*1000  
**G**=1024\*1024\*1024

and so on for **T** ([terabytes](http://www.computerhope.com/jargon/t/terabyte.htm)), **P** ([petabytes](http://www.computerhope.com/jargon/p/petabyte.htm)), **E** ([exabytes](http://www.computerhope.com/jargon/e/exabyte.htm)), **Z** ([zettabytes](http://www.computerhope.com/jargon/z/zettabyt.htm)), and **Y** ([yottabytes](http://www.computerhope.com/jargon/y/yottabyt.htm)).

**Data transfer forms of dd**

|  |  |
| --- | --- |
| blocks=$(isosize -d 2048 /dev/sr0) | Creates an ISO disk image from a CD-ROM, DVD or blue-ray disk.[7] |
| dd if=/dev/sr0 of=isoimage.iso bs=2048 count=$blocks status=progress |
|  |
| [dd if=system.img of=/dev/sdc bs=4096 conv=noerror](https://en.wikipedia.org/wiki/dev/sdc) | Restores a hard disk drive (or an SD card, for example) from a previously created image. |
|
| dd if=/dev/sda2 of=/dev/sdb2 bs=4096 conv=noerror | Clones one partition to another. |
|
| dd if=/dev/ad0 of=/dev/ad1 bs=1M conv=noerror | Clones a hard disk drive "ad0" to "ad1". |
|

**Master boot record backup and restore**

It is possible to repair a master boot record. It can be transferred to and from a repair file.

To duplicate the first two sectors of a floppy drive:

dd if=/dev/fd0 of=MBRboot.img bs=512 count=2

To create an image of the entire x86 master boot record (including a MS-DOS partition table and MBR magic bytes):

dd if=/dev/sda of=MBR.img bs=512 count=1

To create an image of only the boot code of the master boot record (without the partition table and without the magic bytes required for booting):

dd if=/dev/sda of=MBR\_boot.img bs=446 count=1

**Data modification**

dd can modify data in place. For example, this overwrites the first 512 bytes of a file with null bytes:

dd if=/dev/zero of=path/to/file bs=512 count=1 conv=notrunc

The notrunc conversion option means do not truncate the output file — that is, if the output file already exists, just replace the specified bytes and leave the rest of the output file alone. Without this option, dd would create an output file 512 bytes long.

To duplicate a disk partition as a disk image file on a different partition:

dd if=/dev/sdb2 of=partition.image bs=4096 conv=noerror

**Disk wipe**

For security reasons, it is sometimes necessary to have a disk wipe of a discarded device.

To wipe a disk by writing zeros to it, dd can be used this way:

dd if=/dev/zero of=/dev/sda bs=16M

Another approach could be to wipe a disk by writing random data to it:

dd if=/dev/urandom of=/dev/sda bs=16M

When compared to the data modification example above, notrunc conversion option is not required as it has no effect when the dd's output file is a block device.[8]

The bs=16M option makes dd read and write 16 Mebibytes at a time. For modern systems, an even greater block size may be faster. Note that filling the drive with random data may take longer than zeroing the drive, because the random data must be created by the CPU, while creating zeroes is very fast. On modern hard-disk drives, zeroing the drive will render most data it contains permanently irrecoverable.[9] However, with other kinds of drives such as flash memories, much data may still be recoverable by special laboratory techniques.[citation needed]

**Data recovery**

The early history of open-source software for data recovery and restoration of files, drives and partitions included the GNU dd.

**Benchmarking drive performance**

To make drive benchmark test and analyze the sequential (and usually single-threaded) system read and write performance for 1024-byte blocks:

dd if=/dev/zero bs=1024 count=1000000 of=file\_1GB

dd if=file\_1GB of=/dev/null bs=1024

**Generating a file with random data**

To make a file of 100 random bytes using the kernel random driver:

dd if=/dev/urandom of=myrandom bs=100 count=1

**Converting a file to upper case**

To convert a file to uppercase:

dd if=filename of=filename1 conv=ucase,notrunc

**dd examples**

Caution: Use **dd** cautiously — improper usage or entering the wrong values could inadvertently wipe, destroy, or overwrite the data on your hard drive.

dd if=/dev/sr0 of=/home/hope/exampleCD.iso bs=2048 conv=noerror,sync

Create a ISO disk image from the CD in the computer.

dd if=/dev/sdc of=~/disk1.img

Create an img file of the /dev/sda hard drive. To restore that image type: **dd if=disk1.img of=/dev/sdc**

dd if=/dev/sda of=/dev/sdb

Copy the contents from the **if=** drive **/dev/sdc** to the **of=** drive **/dev/sdb**.

1. **Rsync**

## About rsync

Faster, flexible replacement for rcp.

## Description

**rsync** is a fast and extraordinarily versatile file copying tool. It can copy [locally](http://www.computerhope.com/jargon/l/local.htm), to/from another [host](http://www.computerhope.com/jargon/h/hostcomp.htm) over any remote [shell](http://www.computerhope.com/jargon/s/shell.htm), or to/from a remote **rsync** [daemon](http://www.computerhope.com/jargon/d/daemon.htm). It offers a large number of options that control every aspect of its behavior and permit very flexible specification of the set of files to be copied. It is famous for its delta-transfer algorithm, which reduces the amount of data sent over the network by sending only the differences between the source files and the existing files in the destination. **rsync** is widely used for backups and [mirroring](http://www.computerhope.com/jargon/m/mirrorin.htm) and as an improved copy command for everyday use.

**rsync** finds files that need to be transferred using a "quick check" algorithm (by default) that looks for files that have changed in size or in last-modified time. Any changes in the other preserved [attributes](http://www.computerhope.com/jargon/a/attribut.htm) (as requested by options) are made on the destination file directly when the quick check indicates that the file’s data does not need to be updated.

Some of the additional features of **rsync** are:

* Support for copying [links](http://www.computerhope.com/unix/link.htm), [devices](http://www.computerhope.com/jargon/d/device.htm), [owners](http://www.computerhope.com/jargon/o/owner.htm), [groups](http://www.computerhope.com/jargon/g/group.htm), and [permissions](http://www.computerhope.com/jargon/p/permissi.htm)
* **Exclude** and **exclude-from** options similar to [GNU](http://www.computerhope.com/jargon/g/gnu.htm) [tar](http://www.computerhope.com/unix/utar.htm)
* A [CVS](http://www.computerhope.com/jargon/c/cvs.htm) exclude mode for ignoring the same files that CVS would ignore
* Can use any transparent remote shell, including [ssh](http://www.computerhope.com/jargon/s/ssh.htm) or [rsh](http://www.computerhope.com/unix/ursh.htm)
* Does not require super-user privileges
* Pipelining of file transfers to minimize [latency](http://www.computerhope.com/jargon/l/latency.htm) costs
* Support for [anonymous](http://www.computerhope.com/jargon/a/anonymou.htm) or [authenticated](http://www.computerhope.com/jargon/a/auth.htm) **rsync** daemons (ideal for mirroring)

## rsync syntax

Local use:

rsync [*OPTION*...] *SRC*... [*DEST*]

Access via remote shell (**PULL**):

rsync [*OPTION*...] [*USER*@]*HOST*:*SRC*... [*DEST*]

Access via remote shell (**PUSH**):

rsync [*OPTION*...] *SRC*... [*USER*@]*HOST*:*DEST*

Access via **rsync** daemon (**PULL**):

rsync [*OPTION*...] [*USER*@]*HOST*::*SRC*... [*DEST*]

rsync [*OPTION*...] rsync://[*USER*@]*HOST*[:*PORT*]/*SRC*... [*DEST*]

Access via rsync daemon (**PUSH**):

rsync [*OPTION*...] *SRC*... [*USER*@]*HOST*::*DEST*

rsync [*OPTION*...] *SRC*... rsync://[*USER*@]*HOST*[:*PORT*]/*DEST*

Visit man pages for more details on options.

Usefull links:

10 Practical Examples of Rsync Command in Linux

<http://www.tecmint.com/rsync-local-remote-file-synchronization-commands/>

More details about rsync options:

<http://www.computerhope.com/unix/rsync.htm>

rsync examples

***rsync -t \*.htm hope:public\_html/*** 🡪 Use rsync to transfer all .htm files to the public\_html directory on the system named hope.

1. **Find and locate**

**locate(1**) has only one big advantage over find(1): speed.

**find(1),** though, has *many* advantages over locate(1):

* find(1) is primordial, [going back to the very first version of AT&T Unix](http://cm.bell-labs.co/who/dmr/man12.pdf). You will even find it in cut-down embedded Linuxes [via Busybox](http://www.busybox.net/downloads/BusyBox.html#find). It is all but universal.

locate(1) is much younger and [nonstandard](http://pubs.opengroup.org/onlinepubs/9699919799/idx/utilities.html).

The earliest ancestor of locate(1) [appeared in 1983](http://www.eecs.berkeley.edu/Pubs/TechRpts/1983/CSD-83-148.pdf), but it wasn't widely shipped as "locate" [until 1990](http://git.savannah.gnu.org/cgit/findutils.git/tree/ChangeLog). BSD didn't adopt it [until 1994, in 4.4BSD](http://www.freebsd.org/cgi/man.cgi?query=locate&sektion=1).

There are three main flavors of locate(1) in the wild today. In increasing order of features, they are [BSD locate](http://www.freebsd.org/cgi/man.cgi?query=locate&sektion=1), [mlocate](https://fedorahosted.org/mlocate/), and [GNU locate](http://www.gnu.org/software/findutils/). [The BSDs](https://en.wikipedia.org/wiki/Berkeley_Software_Distribution) and Mac OS X ship BSD locate. Most Linuxes ship GNU locate, but Red Hat Linuxes and Arch ship mlocate instead.

Recent versions of GNU locate and mlocate implement all BSD locate options. mlocate implements 6 additional options not in BSD locate: -b, -e, -P, -q, --regex and -w. GNU locate implements those six plus another *four*: -A, -D, -E, and -p. (I'm ignoring aliases and minor differences like -? vs -h vs --help.)

The [big iron](https://en.wikipedia.org/wiki/Big_iron) Unixes don't ship any implementation of locate. (See the man page indexes for [Solaris](http://docs.oracle.com/cd/E26502_01/html/E29030/), [AIX](http://publib16.boulder.ibm.com/pseries/en_US/cmds/aixcmds3/aixcmds3.pdf) and [HP-UX](http://h20566.www2.hp.com/portal/site/hpsc/template.BINARYPORTLET/public/kb/docDisplay/resource.process/?spf_p.tpst=kbDocDisplay_ws_BI&spf_p.rid_kbDocDisplay=docDisplayResURL&spf_p.rst_kbDocDisplay=wsrp-resourceState%3DdocId%253Demr_na-c03231920-1).)

Bottom line, you can't count on locate to be available on all machines, and where present, you can't count on consistent behavior between implementations.

* find(1) has a powerful expression syntax, with many functions, [Boolean operators](http://en.wikipedia.org/wiki/Boolean_expression), etc.
* find(1) can select files by more than just name. It can select by:
  + age
  + size
  + owner
  + file type
  + timestamp
  + permissions
  + depth within the subtree...
* When finding files by name, you can search using [file globbing syntax](http://unix.stackexchange.com/questions/57957/why-do-regular-expressions-differ-from-that-used-to-filter-files/57958#57958) in all versions of find(1), or in GNU or BSD versions, using [regular expressions](http://en.wikipedia.org/wiki/Regex).

Current versions of locate(1) accept glob patterns as find does, but BSD locate doesn't do regexes at all. If you're like me and have to use a variety of machine types, you find yourself preferring grep filtering to developing a dependence on -r or --regex.

locate needs strong filtering more than find does because...

* find(1) doesn't necessarily search the entire filesystem. You typically point it at a subdirectory, a parent containing all the files you want it to operate on. locate(1) simply spews up everything[\*] it knows, leaving it to grep filtering and such to cut its eruption down to size.
* find(1) can *do things* to files it finds, in addition to just finding them. The most powerful and widely supported such operator is -exec, but there are others. In recent GNU and BSD find implementations, for example, you have the -delete and -execdir operators.
* find(1) runs in real time, so its output is always up to date.

Because locate(1) relies on a database updated hours or days in the past, its output can be outdated. (This is the [stale cache problem](https://en.wikipedia.org/wiki/Cache_%28computing%29#Writing_policies).) This coin has two sides:

* + locate can name files that no longer exist.

GNU locate and mlocate have the -e flag to make it check for file existence before printing out the name of each file it discovered in the past, but this eats away some of the locate speed advantage, and isn't available in BSD locate besides.

* + locate will fail to name files that were created since the last database update.

You learn to be somewhat distrustful of locate output, knowing it may be wrong.

* find(1) never has any more privilege than the user running it.

Because locate provides a global service to all users on a system, it wants to have its updatedb process run as root so it can see the entire filesystem. This leads to a choice of security problems:

* + Run updatedb as root, but make its output file world-readable so locate can run without special privileges. This effectively exposes the names of all files in the system to all users. This may be enough of a security breach to cause a real problem.

BSD locate is configured this way on Mac OS X and FreeBSD.

* + Write the database as readable only by root, and make locate [setuid root](http://en.wikipedia.org/wiki/Setuid#setuid_on_executables) so it can read the database. This means locate effectively has to reimplement the OS's permission system so it doesn't show you files you can't normally see. It also increases the [attack surface](http://en.wikipedia.org/wiki/Attack_surface) of your system, specifically risking a [root escalation](https://en.wikipedia.org/wiki/Privilege_escalation) attack.
  + Create a special "locate" user or group to own the database file, and mark the locate binary as setuid/setgid for that user/group so it can read the database. This doesn't prevent privilege escalation attacks by itself, but it greatly mitigates the damage one could cause.

mlocate is configured this way on [Red Hat Enterprise Linux](https://en.wikipedia.org/wiki/Rhel).

You still have a problem, though, because if you can use a debugger on locate or cause it to [dump core](https://en.wikipedia.org/wiki/Core_dump) you can get at privileged parts of the database.

I don't see a way to create a truly "secure" locate command, short of running it separately for each user on the system, which negates much of its advantage over find(1).

Bottom line, both are very useful. locate(1) is better when you're just trying to find a particular file by name, which you know exists, but you just don't remember where it is exactly. find(1) is better when you have a focused area to examine, or when you need any of its many advantages.

Find examples:

<http://www.binarytides.com/linux-find-command-examples/>

Find syntax:

***find [path...] [expression]***

Locate documentation:

<http://www.tutorialspoint.com/unix_commands/locate.htm>

Locate syntax:

***locate [OPTION]... PATTERN...***