[Linux Commands](http://www.bogotobogo.com/DevOps/DevOps-Sys-Admin-Interview-Questions-Commands.php)

tee

Why

sudo echo "deb http://pkg.jenkins-ci.org/debian binary/" >> /etc/apt/sources.list.d/jenkins.list

does not work?

The issue is that it's the shell that handles redirection; it's trying to open the file with user's permissions not those of the process we're running under **sudo**.

So, we should do the following:

$ echo "deb http://pkg.jenkins-ci.org/debian binary/" | sudo tee -a /etc/apt/sources.list.d/jenkins.list

There is a way of doing the same thing not using "tee":

$ sudo sh -c "echo deb http://pkg.jenkins-ci.org/debian binary/ > /etc/apt/sources.list.d/jenkins.list"

Similarly, we can do this when we want to update package list on Debian:

$ echo "deb http://repo.mongodb.org/apt/ubuntu "$(lsb\_release -sc)"/mongodb-org/3.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.0.list

awk

The **awk** is most useful when handling text files that are formatted in a predictable way. For instance, it is excellent at parsing and manipulating tabular data. It operates on a line-by-line basis and iterates through the entire file.

The syntax looks like this:

awk '/search\_pattern/ { action\_to\_take\_on\_matches; another\_action; }' file\_to\_parse

In its simplest form, we can use **awk** like **cat** to simply print all lines of a text file out to the screen. Let's print out our server's fstab file, which lists the filesystems that it knows about:

$ awk '{print}' /etc/fstab

# /etc/fstab: static file system information.

#

# Use 'blkid' to print the universally unique identifier for a

# device; this may be used with UUID= as a more robust way to name devices

# that works even if disks are added and removed. See fstab(5).

#

# <file system> <mount point> <type> <options> <dump> <pass>

# / was on /dev/sda2 during installation

UUID=dca7dd33-7ac2-4e12-a85f-555008cdb302 / ext4 errors=remount-ro 0 1

# /boot/efi was on /dev/sda1 during installation

UUID=D790-E415 /boot/efi vfat defaults 0 1

# swap was on /dev/sda3 during installation

UUID=9555d31f-54fa-4f68-b597-3c0c77335182 none swap sw 0 0

This isn't that useful. Let's try out **awk**'s **search filtering** capabilities:

$ awk '/UUID/' /etc/fstab

# device; this may be used with UUID= as a more robust way to name devices

UUID=dca7dd33-7ac2-4e12-a85f-555008cdb302 / ext4 errors=remount-ro 0 1

UUID=D790-E415 /boot/efi vfat defaults 0 1

UUID=9555d31f-54fa-4f68-b597-3c0c77335182 none swap sw 0 0

Now, **awk** only prints the lines that have "UUID" in them. We can get rid of the extraneous comment line by specifying that UUID must be located at the very beginning of the line:

$ awk '/^UUID/' /etc/fstab

UUID=dca7dd33-7ac2-4e12-a85f-555008cdb302 / ext4 errors=remount-ro 0 1

UUID=D790-E415 /boot/efi vfat defaults 0 1

UUID=9555d31f-54fa-4f68-b597-3c0c77335182 none swap sw 0 0

We can use the **action** section to specify which pieces of information we want to print. For instance, to print only the first column, we can type:

$ awk '/^UUID/ {print $1;}' /etc/fstab

UUID=dca7dd33-7ac2-4e12-a85f-555008cdb302

UUID=D790-E415

UUID=9555d31f-54fa-4f68-b597-3c0c77335182

We can reference every column (as delimited by white space) by variables associated with their column number. The first column can be referenced by **$1** for instance. The entire line can by referenced by **$0**.

tr

We can use **tr** for translating, or deleting, or squeezing repeated characters. It will read from STDIN and write to STDOUT.

lower case to upper case:

$ tr a-z A-Z

abcdef

ABCDEF

(abcdef)

(ABCDEF)

()->{}:

$ tr '()' '{}'

(abcdef)

{abcdef}

It can work with files:

$ cat ok1

(abcdef)

$ tr '()' '{}' < ok1 > ok2

$ cat ok2

{abcdef}

stream editor - Replacing a String

**sed** is a stream editor, we can use it to replacing a string with another either in-place or to a new file.

Suppose we have the following file (t.txt):

12345-abcde

We can in-place (via the option '-i') replace '-' with ':'

$ sed -i 's/-/:/g' t.txt

$ cat t.txt

12345:abcde

Or we can create another file:

$ sed 's/:/-/g' t.txt > tnew.txt

$ cat tnew.txt

12345-abcde

cut

The command **cut** is used for text processing. We can use this command to extract portion of text from a file by selecting columns.

cut OPTION... [FILE]...

In the example below, the option **-f** specifies which field we want to extract, and the option **-d** specifies what is the field delimiter that is used in the input file.

The example displays only the first field of each lines from **/etc/passwd** file using the field delimiter **: (colon)**. In this case, the 1st field is the username.

$ cut -f1 -d':' /etc/passwd

root

daemon

bin

sys

sync

games

man

...

Actually, we can drop the quote in -d:

$ cut -f1 -d**:** /etc/passwd

To check list of groups, we can use the "cut" with "sort":

$ cut -d: -f1 /etc/group | sort

adm

ans

audio

...

voice

whoopsie

www-data

tac

**tac** (which is "cat" backwards) outputs in reverse:

$ cat ok

1

2

3

$ tac ok

3

2

1

tail

$ cat ok

1

2

3

4

5

6

7

8

9

$ tail ok -n 3

7

8

9

watch

We can execute a command periodically using **watch** command.

We can use **watch** to display available disk space (repeat every 10 seconds). By default **watch** command uses 2 second interval, we can change it using **-n** option. The following example executes **df -h** command every 10 seconds.

$ watch -n 10 df -h

Every 10.0s: df -h Sat Aug 22 21:59:35 2015

Filesystem Size Used Avail Use% Mounted on

/dev/sda2 455G 194G 239G 45% /

none 4.0K 0 4.0K 0% /sys/fs/cgroup

udev 1.8G 4.0K 1.8G 1% /dev

tmpfs 355M 1.3M 354M 1% /run

none 5.0M 0 5.0M 0% /run/lock

none 1.8G 41M 1.7G 3% /run/shm

none 100M 60K 100M 1% /run/user

/dev/sda1 511M 3.4M 508M 1% /boot/efi

Symbolic link : ln -s

Symbolic links is a file-system object that points to another file system object (file/directory).

Since a symbolic link refers to a filename, rather than an i-node number, it can be used to link to a file in a different file system. Symbolic links also do not suffer the other limitation of hard links: we can create symbolic links to directories.

$ cat target

This is the target

$ ln -s target target-alias

$ cat target-alias

This is the target

If we try to set a symbolic link with an existing alias, we get the following error:

$ ln -s target target-alias

ln: failed to create symbolic link 'target-alias': File exists

In this case, we can use this:

$ ls -la target\*

-rw-rw-r-- 1 k k 19 Aug 22 22:59 target

lrwxrwxrwx 1 k k 6 Aug 22 23:00 target-alias -> target

$ ln -sf target target-alias

$ ls -la target\*

-rw-rw-r-- 1 k k 19 Aug 22 22:59 target

lrwxrwxrwx 1 k k 6 Aug 22 23:07 target-alias -> target

uptime - load average

We can get the load average from commands like **top** or **uptime**:

k@laptop:~$ uptime

16:48:25 up 32 min, 2 users, load average: 0.58, 1.13, 2.46

From left to right, these numbers show us the average load over the last 1 minute, the last 5 minutes, and the last 15 minutes. In other words, the above output indicates:

load average over the last 1 minute: 0.58

load average over the last 5 minutes: 1.13

load average over the last 15 minutes: 2.46

Assuming 1 cpu machine, it means:

load average over the last 1 minute: 0.58 => The CPU idled for 42% of the time

load average over the last 5 minutes: 1.13 => .13 processes were waiting for the CPU

load average over the last 15 minutes: 2.46 => On average, 1.46 processes were waiting for the CPU

Actually, if the machine has 2 CPUs, then it would mean:

load average over the last 1 minute: 0.58 => The CPU idled for 142% of the time

load average over the last 5 minutes: 1.13 => .87 processes were waiting for the CPU

load average over the last 15 minutes: 2.46 => On average, 0.46 processes were waiting for the CPU

/etc/init.d

The **/etc/init.d** directory contains a number of start/stop scripts for various services on our system.

There are times when we need to start or stop a process cleanly and without using the kill or killall commands. That is where the **/etc/init.d** directory comes in handy.

In order to control any of the scripts in **init.d** manually, we have to have root (or sudo) access. Each script will be run as a command and the structure of the command will look like:

/etc/init.d/command OPTION

Where command is the actual command to run and OPTION can be one of the following:

start

stop

reload

restart

force-reload

Most often we will use either start, stop, or restart. So if we want to stop our network we can issue the command:

/etc/init.d/networking stop

What is inode?

**inode** is a "database" of all file information that tells about file structure. The inode of each file uses a pointer to point to the specific file, directory or object. The pointer is a unique number which usually is referred to as the inode number. For example, to get a listing of an inode number, use the following command:

$ ls -i myfile

19147527 myfile

To get more information that just the inode number, we can use the "stat" command:

$ stat myfile

File: 'myfile'

Size: 11 Blocks: 8 IO Block: 4096 regular file

Device: 802h/2050d Inode: 19147527 Links: 1

Access: (0664/-rw-rw-r--) Uid: ( 1000/ k) Gid: ( 1000/ k)

Access: 2015-10-14 18:42:09.111527150 -0700

Modify: 2015-10-14 18:42:12.703534639 -0700

Change: 2015-10-14 18:42:12.743534723 -0700

Birth: -

We can see that the inode number for the myfile file is 19147527 for both commands.

When an application needs a file, the application exchanges the file name for the inode number from the directory listing. After that, the application uses the inode number for a reference to the file.

To find the inode numbers, we can use the "tree" command with "inode":

$ tree /etc -L 1 --inodes

/etc

|- [3932166] acpi

|- [3932292] adduser.conf

|- [3932167] alternatives

|- [3932293] anacrontab

|- [3938960] ansible

|- [3934719] apache2

What is the difference between a symbolic link and a hard link?

The hard link creates another file with a link to the **same underlying inode** while a soft link is a link to **another inode** in the file system.

Let's create two files: one for symbolic link and the other one for hard link:

$ touch soft; touch hard

Make links:

$ ln hard hard-link

$ ln -s soft soft-link

As we discussed in the previous section, a file is represented by inodes (a pointer to an inode). So, let's check inode:

$ ls -i

19924730 hard 19924730 hard-link 19924727 soft 19924856 soft-link

As we can see from the output, the hard link then creates another file with a link to the same underlying inode (inode number 19924730) while a soft link is a link to another name in the file system.

Want to create the second links?

$ ls -i

19924730 hard 19924730 hard-link 19924730 hard-link2

19924727 soft 19924856 soft-link 19924945 soft-linke2

$ ls -la | awk -F" " '{print $9 $10 $11;}'

hard

hard-link

hard-link2

soft

soft-link->soft

soft-linke2->soft

The hard link then just creates another file with a link to the same inode!

Let's rename the original links:

$ ls -i

19924730 hard-new 19924730 hard-link 19924730 hard-link2

19924727 soft-new 19924856 soft-link 19924945 soft-linke2

So, rename did not have any effects?

Well, we can see the soft links are pointing to a file that's no longer exists!

$ ls -la | awk -F" " '{print $9 $10 $11;}'

hard-link

hard-link2

hard-new

soft-link->soft

soft-linke2->soft

soft-new

Let's see what's happening we try to work on each file.

$ cat hard-link

hard file

$ cat soft-link2

cat: soft-link2: No such file or directory

Once a hard link has been made, the link is pointing to the inode. Renaming the original file will not affect the hard link. Any changes to the data on the inode is reflected in all files that refer to that inode. But soft link become a link to a non-existing file.

Want to see the effects of deletion on hard-link & hard-link2?

$ ls

hard-link hard-link2 hard-new soft-link soft-linke2 soft-new

$ rm soft-new hard-new

$ ls

hard-link hard-link2 soft-link soft-linke2

$ ls -i

19924730 hard-link 19924856 soft-link

19924730 hard-link2 19924945 soft-linke2

The symbolic links are pointing to non-existent file. However, hard-links are still valid:

$ cat hard-link

hard file

A major advantage of a symbolic link is that it can point to a file that's not existing. This is useful if we need a link to a file that is periodically removed and recreated.

Let's create a new soft file with the same name that the link is pointing:

$ echo "soft file" > soft

$ ls

hard-link hard-link2 soft soft-link soft-linke2

$ cat soft-link

soft file

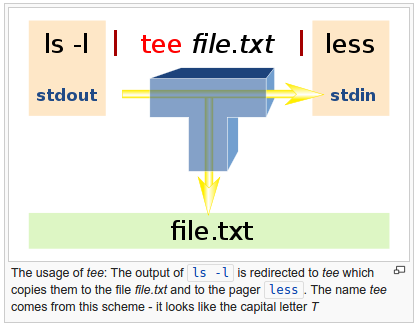
As a practical example, a **symbolic link** could point to a file that gets checked in and out under a version control system, "a.o" file that is re-created by the compiler each time we run make.

**Hard-links** can be very useful when you want to have the same contents (and same permissions!) in several files on the same filesystem.

Take for example a package manager, that creates a /usr/share/doc/$packagename directory for each package that is installed and inside that directory a file called LICENSE with the license information of the package. Many packages on a typical Linux system are GPL licensed, so instead of having 200 copies of the GPL on the filesystem there could be only one copy and 199 links. - from [Linux: Best-practices for Hard Linking?](http://serverfault.com/questions/122364/linux-best-practices-for-hard-linking).

How to redirect output to a file and stdout

**tee** reads standard streams and writes it to both standard output and one or more files, effectively duplicating its input. It is primarily used in conjunction with pipes and filters. The command is named after the T-splitter used in plumbing.



Picture source: <https://en.wikipedia.org/wiki/Tee_(command)>

How to use 'dd' command?

The command **dd** is used to **convert and copy files**.

By default, **dd** reads from stdin and writes to stdout, but these can be changed by using the **if (input file)** and **of (output file)** options.

Backing up a disk to an image will be faster than copying the exact data. Also, disk image make the restoration much more easier. We can create a compressed disk image using **dd**:

$ sudo dd if=/dev/sda1 | gzip >/tmp/sda1disk.img.gz

We often use **dd** to simulate CPU load by filling a file with random content:

$ dd if=/dev/urandom of=500MBfile bs=1M count=500

500+0 records in

500+0 records out

524288000 bytes (524 MB) copied, 103.075 s, 5.1 MB/s

For this, **/dev/urandom** will supply random numbers, which will be generated by the kernel. This will lead to an increased load on the CPU (**sy** - system time). At the same time, the **vmstat** executing in parallel will indicate that between 51% and 55% of the CPU time is being used for the execution of kernel code (for the generation of random numbers, in this case):

$ vmstat 1 5

procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----

r b swpd free buff cache si so bi bo in cs us sy id wa st

4 0 870968 164752 2556 440300 10 17 111 117 289 269 28 10 61 2 0

3 0 870964 158576 2556 445440 0 0 0 0 2341 8240 19 53 28 0 0

6 0 870960 154900 2556 450528 0 0 28 0 2022 6690 19 51 30 0 0

3 1 870960 147560 2676 455800 0 0 232 8 2071 9254 20 55 25 0 0

3 1 870956 142108 2824 462736 0 0 920 8580 1747 5959 15 55 28 2 0

As a high IO read load example, a large file (such as an ISO file) will be read and written to /dev/null using **dd**:

$ dd if=bigfile.iso of=/dev/null bs=1M

While reading, open another terminal and can execute **vmstat** in parallel. It will show the increased IO read load (the **bi** value - blocks received from a block device (blocks/s)):

$ vmstat 1 5

procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----

r b swpd free buff cache si so bi bo in cs us sy id wa st

2 2 882580 146512 44096 540636 14 17 123 115 48 461 28 10 60 1 0

8 1 886024 142780 43864 547788 0 3448 48508 3448 3509 6549 18 18 25 38 0

4 1 886028 101204 43864 590696 32 20 77344 20 2110 4447 10 19 40 32 0

2 1 888788 110368 43784 585296 0 2760 52612 2968 3386 7110 17 18 24 41 0

2 2 891308 110732 43784 591972 64 2528 85440 2528 2986 7273 18 24 32 25 0

Checking linux distribution / OS

To check linux distribution:

$ cat /etc/issue

Ubuntu 14.04.3 LTS \n \l

Or:

$ lsb\_release -a

No LSB modules are available.

Distributor ID: Ubuntu

Description: Ubuntu 14.04.3 LTS

Release: 14.04

Codename: trusty

$ cat /etc/lsb-release

DISTRIB\_ID=Ubuntu

DISTRIB\_RELEASE=14.04

DISTRIB\_CODENAME=trusty

DISTRIB\_DESCRIPTION="Ubuntu 14.04.3 LTS"

$ cat /etc/\*-release

CentOS Linux release 7.2.1511 (Core)

NAME="CentOS Linux"

VERSION="7 (Core)"

ID="centos"

ID\_LIKE="rhel fedora"

VERSION\_ID="7"

PRETTY\_NAME="CentOS Linux 7 (Core)"

ANSI\_COLOR="0;31"

CPE\_NAME="cpe:/o:centos:centos:7"

HOME\_URL="https://www.centos.org/"

BUG\_REPORT\_URL="https://bugs.centos.org/"

CENTOS\_MANTISBT\_PROJECT="CentOS-7"

CENTOS\_MANTISBT\_PROJECT\_VERSION="7"

REDHAT\_SUPPORT\_PRODUCT="centos"

REDHAT\_SUPPORT\_PRODUCT\_VERSION="7"

CentOS Linux release 7.2.1511 (Core)

CentOS Linux release 7.2.1511 (Core)

Or:

$ cat /etc/os-release

NAME="Ubuntu"

VERSION="14.04.3 LTS, Trusty Tahr"

ID=ubuntu

ID\_LIKE=debian

PRETTY\_NAME="Ubuntu 14.04.3 LTS"

VERSION\_ID="14.04"

HOME\_URL="http://www.ubuntu.com/"

SUPPORT\_URL="http://help.ubuntu.com/"

BUG\_REPORT\_URL="http://bugs.launchpad.net/ubuntu/"

How to setup passwordless `sudo` on Linux?

We can add a user to a **wheel** group:

$ sudo usermod -g wheel my-user-name

"The term **wheel** refers to a user account with a wheel bit, a system setting that provides additional special system privileges that empower a user to execute restricted commands that ordinary user accounts cannot access. The term is derived from the slang phrase big wheel, referring to a person with great power or influence." - [Wheel (Unix term)](https://en.wikipedia.org/wiki/Wheel_(Unix_term))

We may need to uncomment a line using **sudo visudo** and it should look like this:

## Allows people in group wheel to run all commands

%wheel ALL=(ALL) ALL

## Same thing without a password

%wheel ALL=(ALL) NOPASSWD: ALL

usermod & gpasswd

We can add a user (for example, **sfvue**) to a group (**apache**):

$ sudo usermod -a -G apache sfvue

$ groups sfvue

sfvue : wheel apache

To remove a user from a group:

$ sudo gpasswd -d sfvue apache

$ groups sfvue

sfvue : wheel

Also check [Managing User Account](http://www.bogotobogo.com/Linux/linux_user_account_useradd_usermod_userdel.php)