# SED

1. To change info to debug in line after com.osmosix

<logger name="com.osmosix">

<level value="info" />

sed '/<logger name="com.osmosix">/!b;n;c<level value="degbug" \/>' log4j.xml > log4j

Note the two n's, it replaces after two lines and so forth.

# Random password generator

PASSWDDB="$(openssl rand -base64 12)"

# Bashrc

alias wpinstall="curl -L -o 'wp.sh' https://gist.githubusercontent.com/bgallagh3r/2853221/raw/f3e2f2fa3048bbeb6d35799af068965bc5fd9b26/wp.sh && bash wp.sh" then source ~/.bashrc Go into a dir you want to install WP in and just type wpinstall It's that easy!

# RPM

rpm -q sudo > /dev/null && echo sudo is installed || echo sudo NOT installed

Operation

RPM

deb

Install package

rpm -i foo.rpm

dpkg --install foo.deb

Install package, dependencies

yum install foo

apt-get install foo

Remove package

rpm -e foo.rpm

dpkg --remove foo.deb

Remove package, dependencies

yum remove foo

apt-get autoremove foo

Update package

rpm -U foo.rpm

dpkg --install foo.deb

Update package, dependencies

yum update foo

apt-get install foo

Update entire system

yum update

apt-get dist-upgrade

Show all installed packages

rpm -qa

or

dpkg --list

yum list installed

Get information on package

rpm -qil foo

dpkg --listfiles foo

Show packages named

foo

yum list "foo"

apt-cache search foo

Show all available packages

yum list

apt-cache dumpavail foo

What package is

file

part of?

rpm -qf file

dpkg --search file

# Touch

touch a{1..1000}

Example 6:How about generating numbers?. We can generate numbers using flower braces with echo command. The other way to generate numbers is seq command.

echo {1..10}

touch {0..1000}{0..1000}

Example10: How about creating files as multiples of 2.

touch {1..100..2}

option "B" will make file sarath.txt older 50 seconds than satish.txt.

[root@satish test]# touch -r satish.txt B 50 sarath.txt

# mkdir

mkdir -p 2012/{1..12}/{1..30}

# Echo

echo {1..10}

echo ${data//[-.\_]/} -🡪 to remove dot from string

# locate

|  |  |
| --- | --- |
| [set] | Matches any character in the set of characters, for example [adf] will match any occurrence of "a", "d", or "f" |
| [!set] | Matches any character not in the set of characters |

Eg locate bas[a]

# If

if (which yum >/dev/null 2>&1); then echo "exists"; else echo "doesnt"; fi

# Find

Searching only for directories named "gcc":  
$ find /usr -type d -name gcc

Searching only for regular files named "test1":  
$ find /usr -type f -name test1

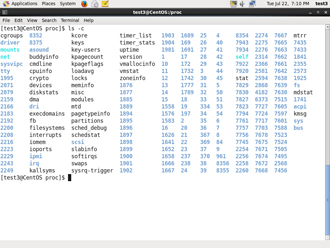
# WGET

wget -r --no-parent --reject "index.html\*" http://192.168.3.248/bundles/installer454/

# Partition

Df –Th

# proc Filesystem

[[](https://d37djvu3ytnwxt.cloudfront.net/assets/courseware/78027794cea494e7bde1fa395445b7ca/asset-v1:LinuxFoundationX+LFS101x+1T2016+type@asset+block/LFS01_ch08_screen12.jpg)](https://d37djvu3ytnwxt.cloudfront.net/assets/courseware/78027794cea494e7bde1fa395445b7ca/asset-v1:LinuxFoundationX+LFS101x+1T2016+type@asset+block/LFS01_ch08_screen12.jpg)Certain filesystems like the one mounted at /proc are called **pseudo filesystems** because they have no permanent presence anywhere on disk.

The /proc filesystem contains virtual files (files that exist only in memory) that permit viewing constantly varying kernel data. This filesystem contains files and directories that mimic kernel structures and configuration information. It doesn't contain real files but runtime system information (e.g. system memory, devices mounted, hardware configuration, etc). Some important files in /proc are:

/proc/cpuinfo  
/proc/interrupts  
/proc/meminfo  
/proc/mounts  
/proc/partitions  
/proc/version

/proc has subdirectories as well, including:

/proc/<Process-ID-#>  
/proc/sys

The first example shows there is a directory for every **process** running on the system which contains vital information about it. The second example shows a virtual directory that contains a lot of information about the entire system, in particular its hardware and configuration. The /proc filesystem is very useful because the information it reports is gathered only as needed and never needs storage on disk.

# boot Directory

The /boot directory contains the few essential files needed to boot the system. For every alternative kernel installed on the system there are four files:

1. vmlinuz: the compressed Linux kernel, required for booting
2. initramfs: the initial ram filesystem, required for booting, sometimes called initrd, not initramfs
3. config: the kernel configuration file, only used for debugging and bookkeeping
4. System.map: kernel symbol table, only used for debugging

Each of these files has a kernel version appended to its name.

The **Grand Unified Bootloader** (**GRUB**) files (such as /boot/grub/grub.conf or /boot/grub2/grub2.cfg) are also found under the /boot directory.

The images show an example listing of the /boot directory, taken from a **CentOS** system that has three installed kernels. Names would vary and things would look somewhat different on a different distribution.

# diff3 and patch

You can compare three files at once using **diff3**, which uses one file as the reference basis for the other two. For example, suppose you and a co-worker both have made modifications to the same file working at the same time independently. **diff3** can show the differences based on the common file you both started with. The syntax for **diff3** is as follows:

$ diff3 MY-FILE COMMON-FILE YOUR-FILE

The graphic shows the use of **diff3**.

Many modifications to source code and configuration files are distributed utilizing **patches**, which are applied, not suprisingly, with the **patch** program. A patch file contains the **deltas** (changes) required to update an older version of a file to the new one. The patch files are actually produced by running **diff** with the correct options, as in:

$ diff -Nur originalfile newfile > patchfile.patch (it includes data from both files)

Distributing just the patch is more concise and efficient than distributing the entire file. For example, if only one line needs to change in a file that contains 1,000 lines, the **patch** file will be just a few lines long.

To apply a patch you can just do either of the two methods below:

$ patch -p1 < patchfile.patch   
$ patch originalfile patchfile

he above example was so simple that it works only with one file. We will see how to create and apply patch for a complete source tree by taking “openvpn” source code as example.

I’ve downloaded 2 version of openvpn, openvpn-2.3.2 and openvpn-2.3.4.

tar -xvzf openvpn-2.3.2.tar.gz

tar -xvzf openvpn-2.3.4.tar.gz

Now we will create the patch using the following command.

diff -Naur /usr/src/openvpn-2.3.2 /usr/src/openvpn-2.3.4 > openvpn.patch

The above command will operate recursively and find the differences, and place those differences in the patch file.

4. Apply Patch File to a Source Code Tree

The following patch commands can be used to apply the patch to source tree.

# patch -p3 < /root/openvpn.patch

patching file openvpn-2.3.2/aclocal.m4

patching file openvpn-2.3.2/build/Makefile.in

patching file openvpn-2.3.2/build/msvc/Makefile.in

...

Please note that we are executing the command from /usr/src/. The patch file contains all the filenames in absolute path format( from root ). So when we execute from /usr/src, without the “-p” option, it will not work properly.

-p3 tells the patch command to skip 3 leading slashes from the filenames present in the patch file. In our case, the filename in patch file is “/usr/src/openvpn-2.3.2/aclocal.m4”, since you have given “-p3”, 3 leading slashes, i.e. until /usr/src/ is ignored.

5. Take a Backup before Applying the Patch using -b

You can take a backup of the original file before applying the patch command using the -b option as shown below.

$ patch -b < hello.patch

patching file hello.c

Now you will have a file name “hello.c.orig”, which is the backup of the original hello.c.

You can also use -V to decide the backup filename format as shown below. Now you will have a file name “hello.c.~1~”.

$ patch -b -V numbered < hello.patch

patching file hello.c

6. Validate the Patch without Applying (Dry-run Patch File)

You can dry run the patch command to see if you are getting any errors, without patching the file using –dry-run option as shown below.

$ patch --dry-run < hello.patch

patching file hello.c

You can see that hello.c is not modified at all.

7. Reverse a Patch that is Already Applied (Undo a Patch)

You can use the -R option to reverse a patch which is applied already.

$ patch < hello.patch

patching file hello.c

$ ls -l hello.c

-rw-r--r-- 1 lakshmanan users 94 2014-10-07 20:05 hello.c

$ patch -R < hello.patch

patching file hello.c

$ ls -l hello.c

-rw-r--r-- 1 lakshmanan users 62 2014-10-07 20:04 hello.c

You can notice from the filesize, that the patch, which is applied already is reversed when we used the -R option.

# Compress Data Using xz

**xz** is the most space efficient compression utility used in Linux and is now used by [www.kernel.org](https://www.kernel.org/) to store archives of the Linux kernel. Once again it trades a slower compression speed for an even higher compression ratio.

Some usage examples:

|  |  |
| --- | --- |
| **Command** | **Usage** |
| $ xz \* | Compress all of the files in the current directory and replace each file with one with a .xz extension. |
| xz foo | Compress the file foo into foo.xz using the default compression level (-6), and remove foo if compression succeeds. |
| xz -dk bar.xz | Decompress bar.xz into  bar and don't remove bar.xz even if decompression is successful. |
| xz -dcf a.txt b.txt.xz > abcd.txt | Decompress a mix of compressed and uncompressed files to standard output, using a single command. |
| $ xz -d \*.xz | Decompress the files compressed using xz. |
| tar xvf mydir.tar | Extract all the files in mydir.tar into the mydir directory |
| $ tar zcvf mydir.tar.gz mydir | Create the archive and compress with gzip |
| $ tar jcvf mydir.tar.bz2 mydir | Create the archive and compress with bz2 |
| $ tar Jcvf mydir.tar.xz mydir | Create the archive and compress with xz |
| $ tar xvf mydir.tar.gz | Extract all the files in mydir.tar.gz into the mydir directory. Note you do not have to tell tar it is in gzip format. |

# Dialog box

dialog --aspect 80 --infobox "Configuring CCM Parameters" 15 50

# tee

The following command (with the help of tee command) writes the output both to the screen (stdout) and to the file.

$ ls | tee file

By default tee command overwrites the file. You can instruct tee command to append to the file using the option –a as shown below.

$ ls | tee –a file

You can also write the output to multiple files as shown below.

$ ls | tee file1 file2 file3

Tee /tmp/test << ‘EOF’

Test

Test

EOF