

## How To search for a string in a file using grep

grep searches the input files for lines containing a match to a given pattern list. When it finds a match in a line, it copies the line to standard output (by default), or does whatever other sort of output you have requested with options.

grep can simply be invoked: **\$ grep 'STRING' filename**

This is OK but it does not show the true power of grep. First this only looks at one file. A cool example of using grep with multiple file would be to find all files in a directory that contains the name of a person. This can be easily accomplished using a grep in the following way :

**\$ grep 'Nikesh J' \***

Notice the use of single quotes; This are not essential but in this example it was required since the name contains a space. Double quotes could also have been used in this example.

Grep Regular Expression

grep can search for complicated pattern to find what you need. Here is a list of some of the special characters used to create a regular expression:

`.' The period `.' matches any single character.

`?' The preceding item is optional and will be matched at most once.

`\*' The preceding item will be matched zero or more times.

`+' The preceding item will be matched one or more times.

So an example of a regular expression search would be: **\$ grep "\<[A-Za-z].\*" file**

This will search for any word which begins with a letter upper or lower case.

For more details check: **\$ man grep**

## Linux file permissions

The basics of file ownership and permissions on Linux. Learn to understand who are the owners of a file or directory, how the file permissions work and how you can view them, and learn how to set basic file permissions yourself.

< **Permissions and ownership - why?** >

If you can't access some of the files on your very own Linux system, it's usually because of misconfigured file access permissions. If you are the only user on your Linux box, you may be wondering what's the point of having all these permissions (or lack thereof) that restrict your access to your very own penguin OS. However, before pulling your hair off, you must keep in mind Linux is designed to be a multi-user environment. In an environment with more than one users, it is crucial to

have a secure system for deciding which files are yours and who can fiddle with them.

Even if you're the only user on an ordinary desktop system, file permissions help keeping your important files safe, both from outsiders and your own mistakes. 😊

### < Understanding file ownership >

Every file on your Linux system, including directories, is owned by a specific user and group. Therefore, file permissions are defined separately for users, groups, and others.

**User:** The username of the person who owns the file. By default, the user who creates the file will become its owner.

**Group:** The usergroup that owns the file. All users who belong into the group that owns the file will have the same access permissions to the file. This is useful if, for example, you have a project that requires a bunch of different users to be able to access certain files, while others can't. In that case, you'll add all the users into the same group, make sure the required files are owned by that group, and set the file's group permissions accordingly.

**Other:** A user who isn't the owner of the file and doesn't belong in the same group the file does. In other words, if you set a permission for the "other" category, it will affect everyone else by default. For this reason, people often talk about setting the "world" permission bit when they mean setting the permissions for "other."

### < Understanding file permissions >

There are three types of access permissions on Linux: read, write, and execute. These permissions are defined separately for the file's owner, group and all other users.

**Read permission.** On a regular file, the read permission bit means the file can be opened and read. On a directory, the read permission means you can list the contents of the directory.

**Write permission.** On a regular file, this means you can modify the file, aka write new data to the file. In the case of a directory, the write permission means you can add, remove, and rename files in the directory. This means that if a file has the write permission bit, you are allowed to modify the file's contents, but you're allowed to rename or delete the file only if the permissions of the file's directory allow you to do so.

**Execute permission.** In the case of a regular file, this means you can execute the file as a program or a shell script. On a directory, the execute permission (also called the "search bit") allows you to access files in the directory and enter it, with the cd command, for example. However, note that although the execute bit lets you enter the directory, you're not allowed to list its contents, unless you also have the read permissions to that directory.

### < How to view file permissions >

You can view the access permissions of a file by doing the long directory listing with the ls -l command. This is what a long directory listing might look like:

```
me@puter: /home/writers$ ls -l
```

```
total 17
```

```
drwxr-xr-x 3 nana writers 80 2005-09-20 21:37 dir
-rw-r----- 1 nana writers 8187 2005-09-19 13:35 file
-rwxr-xr-x 1 nana writers 10348 2005-07-17 20:31 otherfile
```

What does the output of `ls -l` mean? The very first column, the one that looks like a bunch of mumbo jumbo, shows the file type and permissions. The second column shows the number of links (directory entries that refer to the file), the third one shows the owner of the file, and the fourth one shows the group the file belongs to. The other columns show the file's size in bytes, date and time of last modification, and the filename.

The first column, the one that shows the file's permissions and looks like mumbo jumbo, is organized into four separate groups, although it certainly doesn't look very organized.

The first group consists of only one character, and it shows the file's type. For example, `d` means a directory and `-` means a normal file, so if you take a look at our example output, you'll notice `dir` is a directory, while `file` and `otherfile` are regular files.

The first character can be any of these:

*d = directory*  
*- = regular file*  
*l = symbolic link*  
*s = Unix domain socket*  
*p = named pipe*  
*c = character device file*  
*b = block device file*

The next nine characters show the file's permissions, divided into three groups, each consisting of three characters. The first group of three characters shows the read, write, and execute permissions for user, the owner of the file. The next group shows the read, write, and execute permissions for the group of the file. Similarly, the last group of three characters shows the permissions for other, everyone else. In each group, the first character means the read permission, the second one write permission, and the third one execute permission.

The characters are pretty easy to remember.

**r** = read permission  
**w** = write permission  
**x** = execute permission  
**-** = no permission

What does this mean in practice? Let's have an example. Remember the imaginary directory listing we did at the beginning? The output looked like this:

```
drwxr-xr-x 3 nana writers 80 2005-09-20 21:37 dir  
-rw-r----- 1 nana writers 8187 2005-09-19 13:35 file  
-rwxr-xr-x 1 nana writers 10348 2005-07-17 20:31 otherfile
```

As we already noticed, `dir` is a directory, because the first column begins with a `d`. The owner of this directory is user `nana` and the group owner is `writers`. The first three characters, `rwX`, indicate the directory's owner, `nana` in this case, has full access to the directory. The user `nana` is able to access, view, and modify the files in that directory. The next three characters, `r-x`, indicate that all users belonging to group `writers` have read and execute permissions to the directory. They can change into the directory, execute files, and view its contents. However, because they don't have write permissions, they can't make any changes to the directory content. Finally, the last three characters, `r-x`, indicate that all the users who are not `nana` or don't belong into group `writers`, have read and execute permissions in the directory.

How about `file`? Because the first column begins with a `-`, the file is a regular file, owned by user `nana`

and group writers, just like the directory in our example. The first three characters, `rw-`, indicate the owner has read and write access to the file. According to the next three characters, `r-`, the users belonging to group writers can view the file but not modify or execute it. The final three characters, `---`, indicate no one else has any access to the file.

Similarly, you can see `otherfile` is a regular file and its owner has full access to it, while everyone else can read and execute the file but not modify it.

### < **How to set file permissions - symbolic mode** >

You can set file permissions with the `chmod` command. Both the root user and the file's owner can set file permissions. `chmod` has two modes, symbolic and numeric.

The symbolic mode is pretty easy to remember. First, you decide if you set permissions for the user (u), the group (g), others (o), or all of the three (a). Then, you either add a permission (+), remove it (-), or wipe out the previous permissions and add a new one (=). Next, you decide if you set the read permission (r), write permission (w), or execute permission (x). Last, you'll tell `chmod` which file's permissions you want to change.

Let's have a couple of examples. Suppose we have a regular file called `testfile`, and the file has full access permissions for all the groups (long directory listing would show `-rwxrwxrwx` as the file's permissions).

Wipe out all the permissions but add read permission for everybody:

#### **\$ chmod a=r testfile**

After the command, the file's permissions would be `-r--r--r--`

Add execute permissions for group:

#### **\$ chmod g+x testfile**

Now, the file's permissions would be `-r--r-xr--`

Add both write and execute permissions for the file's owner. Note how you can set more than one permission at the same time:

#### **\$ chmod u+wx testfile**

After this, the file permissions will be `-rwxr-xr--`

Remove the execute permission from both the file's owner and group. Note, again, how you can set them both at once:

#### **\$ chmod ug-x testfile**

Now, the permissions are `-rw-r--r--`

As a summary, have a look at this quick reference for setting file permissions in symbolic mode:

<b>Which user?</b>	
u	user/owner
g	group
o	other
a	all

<b>What to do?</b>	
+	add this permission
-	remove this permission
=	set exactly this permission
<b>Which permissions?</b>	
r	read
w	write
x	execute

### < How to set file permissions - numeric mode >

The other mode in which `chmod` can be used is the numeric mode. In the numeric mode, the file permissions aren't represented by characters. Instead, they are represented by a three-digit octal number.

**4** = read (r)

**2** = write (w)

**1** = execute (x)

**0** = no permission (-)

To get the permission bits you want, you add up the numbers accordingly. For example, the `rw` permissions would be  $4+2+1=7$ , `rx` would be  $4+1=5$ , and `rw` would be  $4+2=6$ . Because you set separate permissions for the owner, group, and others, you'll need a three-digit number representing the permissions of all these groups.

Let's have an example.

#### **\$ chmod 755 testfile**

This would change the `testfile`'s permissions to `-rwxr-xr-x`. The owner would have full read, write, and execute permissions ( $7=4+2+1$ ), the group would have read and execute permissions ( $5=4+1$ ), and the world would have the read and execute permissions as well.

Let's have another example:

#### **\$ chmod 640 testfile**

In this case, `testfile`'s permissions would be `-rw-r---`. The owner would have read and write permissions ( $6=4+2$ ), the group would have read permissions only (4), and the others wouldn't have any access permissions (0).

The numeric mode may not be as straightforward as the symbolic mode, but with the numeric mode, you can more quickly and efficiently set the file permissions. This quick reference for setting file permissions in numeric mode might help:

<b>Which number?</b>	
0	—
1	-x

2	-w-
3	-wx
4	r-
5	r-x
6	rw-
7	rwX

## How to change a file's owner and group in Linux

< **chown - change the owner of a file** >

You can change the owner and group of a file or a directory with the chown command. Please, keep in mind you can do this only if you are the root user or the owner of the file.

Set the file's owner:

**\$ chown username somefile**

After giving this command, the new owner of a file called somefile will be the user username. The file's group owner will not change. Instead of a user name, you can also give the user's numeric ID here if you want.

You can also set the file's group at the same time. If the user name is followed by a colon and a group name, the file's group will be changed as well.

**\$ chown username:usergroup somefile**

After giving this command, somefile's new owner would be user username and the group usergroup.

You can set the owner of a directory exactly the same way you set the owner of a file:

**\$ chown username somedir**

Note that after giving this command, only the owner of the directory will change. The owner of the files inside of the directory won't change.

In order to set the ownership of a directory and all the files in that directory, you'll need the -R option:

**\$ chown -R username somedir**

Here, R stands for recursive because this command will recursively change the ownership of directories and their contents. After issuing this example command, the user username will be the owner of the directory somedir, as well as every file in that directory.

Tell what happens:

**\$ chown -v username somefile**

changed ownership of 'somefile' to username

Here, v stands for verbose. If you use the -v option, chown will list what it did (or didn't do) to the file.

The verbose mode is especially useful if you change the ownership of several files at once. For example, this could happen when you do it recursively:

### **\$ chown -Rv username somedir**

*changed ownership of 'somedir/' to username*

*changed ownership of 'somedir/boringfile' to username*

*changed ownership of 'somedir/somefile' to username*

As you can see, chown nicely reports to you what it did to each file.

### **< chgrp - change the group ownership of a file >**

In addition to chown, you can also use the chgrp command to change the group of a file or a directory. You must, again, be either the root user or the owner of the file in order to change the group ownership.

chgrp works pretty much the same way as chown does, except it changes the file's user group instead of the owner, of course.

### **\$ chgrp usergroup somefile**

After issuing this command, the file somefile will be owned by a user group usergroup. Although the file's group has changed to usergroup, the file's owner will still be the same.

The options of using chgrp are the same as using chown. So, for example, the -R and -v options will work with it just like they worked with chown:

### **\$ chgrp -Rv usergroup somedir**

*changed group of 'somedir/' to usergroup*

*changed group of 'somedir/boringfile' to usergroup*

*changed group of 'somedir/somefile' to usergroup*

chown nicely reports to you what it did to each file.

## **HowTo keep your Ubuntu up-to-date**

It is important to have the system updated, so that we have all the latest patches, security fixes and packages upgrades from the repositories.

First, navigate to: System → Administration → Software Sources

and check that all repositories are enabled.

Then, open the terminal: Applications → Accessories → Terminal

and type: (you need to provide the password here)

**sudo apt-get update**

**sudo apt-get upgrade**

**sudo apt-get dist-upgrade**

With this you are done with the system update.

# Limiting Access to Websites/Directories with .htaccess

This post is not so much a tutorial as my own notes on restricting access with .htaccess files and apache. As has been the case with many of my previous tutorials, the basis is writing the steps down so I can refer to them later. Turns out making notes public on a blog is a good idea. In any event, this will outline restricting access to directories on a user-level with .htaccess.

## **Create the .htaccess file**

To limit access to a directory we need to create a .htaccess file where we will outline the restrictions for the location. Any folder within your publicly accessible web page can have its own custom .htaccess file. note: some shared hosting companies do not allow custom .htaccess restrictions for individual sites. You may need to check with your host on this.

Within your .htaccess file you would include something along these lines:

```
# sample .htaccess file
AuthName "Private Website"
AuthType basic
AuthUserFile /path/to/.htpasswd
require user username (optional)
```

---

In the above sample config “Private Website” can be any message you want displayed to the user when trying to authenticate to that page. /path/to/.htpasswd is what we will work on next in generating usernames and hashed passwords for authentication. require user username can limit access to only those users listed.

## **Create the .htpasswd file**

In the .htaccess file we’ve outlined a path/to/.htpasswd file which we need to also create. It is a good idea to keep this file in a non web-accessible location. For example, if your web root is /var/www/html/ you might put the .htpasswd file in /var/www/.htpasswd. This way it is not accessible publicly and limits the chances of someone being able to get a hold of and attempt to break your hashed passwords for access.

To populate the .htpasswd file we’ll use the command htpasswd. To initially create the file we’d use:

```
htpasswd -cm /var/www/.htpasswd user-one
```

The -c will initially create the file. The -m will md5 encrypt the passwords for additional security. The htpasswd command will prompt you for a password.

To add additional users to your .htpasswd access list use:

```
htpasswd -m /var/www/.htpasswd user-two
```

**Be careful not to use the -c option when adding additional users as this will recreate the file and overwrite previous entries.**

Once these two files are in place access to the folder containing the .htaccess file will be limited to only those users listed within the .htpasswd file and require authentication via a password. This is great for sharing web accessible files with only certain users, creating private folders, etc.

If your host allows custom .htaccess file creation but does not provide you access to the htpasswd command you can try to generate your .htpasswd file using an apache installation on a local machine and copying the resulting files over.

## tar command basic operations

The tar (i.e., tape archive) command is used to convert a group of files into an archive (no compress).

Unlike some other archiving programs, and consistent with the Unix philosophy that each individual program should be designed to do only one thing but do it well, tar does not perform compression. However, it is very easy to compress archives created with tar by using specialized compression utilities.

The basic tar syntax is

**tar option(s) archive\_name file\_name(s)**

Remember certain option defined, c for create, z for extract and t for test. v verbose, and f is file. There are two common archive formats that people are interested in, tar.bz2 and tar.gz (tgz). tar.bz2 is more compress than tar.gz, but tar.gz is faster for creating and extracting.

Create and Compress

To create a tar.gz archive, if given list of files, use option z to indicate tar.gz:

**tar czvf filename.tar.gz myfile1 myfile2 myfile3**

To create a tar.gz archive from a folder, if folder must be included in the archive, that means will be extracted with the folder too:

**tar czvf filename.tar.gz foldername/\***

By default, tar creates an archive of copies of the original files and/or directories, and the originals are retained. However, they can be removed when using tar by adding the `--remove-files` option.

Extract and List

To test the tar file to see the content of the tar.gz file, use t option.

**tar tzvf filename.tar.gz**

To extract the tar.gz file, use x option.

**tar xzvf filename.tar.gz**

To extract the tar.gz file to a specified folder

**tar xzvf filename.tar.gz -C foldername/  
tar.bz2**

For tar.bz2, use the same option but change everything from z to j, for example:

**tar cjvf filename.tar.bz2 myfile1 myfile2 myfile3**

**tar cjvf filename.tar.bz2 foldername/\***

**tar tjvf filename.tar.bz2**

**tar xjvf filename.tar.bz2**

# How to install second hard drive in Ubuntu Linux

So you've been using linux for awhile now, and it's time to install another hard drive for some more storage? Maybe it's finally time to wipe out a Window's or NTFS partition.

In any event, I wrote this post so you would have a little help through this step. I'm going to assume that you already know how to install a hard drive. I'm also going to assume that you knew how to make it a master or slave, you've checked to make sure that it shows up in bios, and that it was intalled properly. It also assumes you've already formatted your drive in linux ext3 format, using a tool like gParted, or something similar.

If you've done all of these things, then boot up your system - and let's get going. I'm using Ubuntu linux - formerly Breezy 5.10, but I've recently upgraded to Dapper 6.06.

Open up a terminal window and run the following command:

```
$ sudo fdisk -l
```

You should get a listing of the hard drives installed on your computer. There will be a little paragraph for each one that will look like this:

```
Disk /dev/hda: 40.0 GB, 40020664320 bytes  
255 heads, 63 sectors/track, 4865 cylinders  
Units = cylinders of 16065 * 512 = 8225280 bytes
```

```
Device Boot Start End Blocks Id System  
/dev/hda1 * 1 4678 37576003+ 83 Linux  
/dev/hda2 4679 4865 1502077+ 5 Extended  
/dev/hda5 4679 4865 1502046 82 Linux swap / Solaris
```

In windows disk drives are assigned an alphabet letter, and traditionally - floppy disk drives were a: and / or b:, and the main hard drive was c:/. Then the first cdrom or dvdrom was d:/, and any additional drives would be e:, f:, and so on. In linux it's kind of the same, but in a different format. All hard drives installed are listed in the 'device' or /dev directory. All drives start with the appendage "hd" (I think for 'hard drive'.

So, if you have 2 hard drives and one cdrom - then you have 3 devices. You have a hda, hdb, and hdc. The number of partitions comes next. If your main hard drive is linux - and you have 3 partitions, then you'll have a hda1, hda2, and hda5. The partition numbers aren't in a logical order - hda5 is always the swap partition.

So know, if you see your hard drive listed in the sudo fdisk -l, then you know you can mount it. Your computer has a startup file that tells it what to mount when it boots. We need to edit this file and add the new drive.

But first we have to create a directory. We have to create what's called a "mount point". A mount point is a virtual directory. It's like saying - mount that hard drive from this directory.

The most logical place to create the mount point for the new hard drive is either the /mnt or /media folder. Many would say that /mnt is the only place it should be - the mount folder. However, Ubuntu always mounts all dvd, cd, and removable media in /media. I chose to make my mount point here for just that very reason.

Think of what you want to call the new mount point name. Just make sure you don't use any special characters or spaces in the name. I called mine linuxstore. Now, in terminal run the following command substituting my mount point name for yours:

```
$ sudo mkdir /media/linuxstore
```

Next, here's one of the most important things....and something that I didn't find in any of the articles on the web when I was trying to figure out how to do this. You have to make the mount point directory 'writable'. In other words, you have to give it writable permissions. They have to be world-writable permissions since you aren't a member of the 'root' group in which all mount points are owned.

So, now you want to run the following command (again substituting my mount point name for yours):

```
$ chmod 777 /media/linuxstore
```

If you want to mount your drive right away, and you don't care if it's mounted automatically every time you boot - then in terminal run the following command:

```
$ mount /dev/hdd1 /media/linuxstore
```

There! Now you are temporarily mounted. But...if you want it to be permanent, you need to edit your filesystem tab file. Run the following command in terminal:

```
$ gedit /etc/fstab
```

The text editor window will appear with the fstab file loaded up. You will see something that looks kind of like this:

```
# /etc/fstab: static file system information.  
#  
# proc /proc proc defaults 0 0  
/dev/hda1 / ext3 defaults,errors=remount-ro 0 1  
/dev/hda5 none swap sw 0 0  
/dev/hdb1 /media/hdb1 ext3 defaults 0 0  
/dev/hdc /media/cdrom0 udf,iso9660 user,noauto 0 0  
/dev/fd0 /media/floppy0 auto rw,user,noauto 0 0
```

All you have to do is add a new line for the new drive...

I will add the following line to my fstab for my new drive:

```
/dev/hdd1 /media/linuxstore ext3 defaults 0 0
```

Just be sure to substitute both the name of my hard drive for yours (mine is hdd1, is yours hdc1 or another name?), and my mount point name for yours. Then save the file.

Now you will have the new hard drive mounted and writable both every time you boot. In Ubuntu, you should find your new drive listed under your 'Places' menu. To make the hard drive show up right now, without rebooting - just reload your fstab file with the following command:

```
$ sudo mount -a
```

Now you're done! Enjoy your new storage drive in Linux!

# Howto: Backup and restore your system!

Hi, and welcome to the successful backing-up and restoring of a Linux system!

Most of you have probably used Windows before you started using Ubuntu. During that time you might have needed to backup and restore your system. For Windows you would need proprietary software for which you would have to reboot your machine and boot into a special environment in which you could perform the backing-up/restoring (programs like Norton Ghost).

During that time you might have wondered why it wasn't possible to just add the whole c:\ to a big zip-file. This is impossible because in Windows, there are lots of files you can't copy or overwrite while they are being used, and therefore you needed specialized software to handle this.

Well, I'm here to tell you that those things, just like rebooting, are Windows CrazyThings™. There's no need to use programs like Ghost to create backups of your Ubuntu system (or any Linux system, for that matter). In fact; using Ghost might be a very bad idea if you are using anything but ext2. Ext3, the default Ubuntu partition, is seen by Ghost as a damaged ext2 partition and does a very good job at screwing up your data.

## 1: Backing-up

“What should I use to backup my system then?” might you ask. Easy; the same thing you use to backup/compress everything else; TAR. Unlike Windows, Linux doesn't restrict root access to anything, so you can just throw every single file on a partition in a TAR file!

To do this, become root with

Code:

**sudo su**

and go to the root of your filesystem (we use this in our example, but you can go anywhere you want your backup to end up, including remote or removable drives.)

Code:

**cd /**

Now, below is the full command I would use to make a backup of my system:

Code:

```
tar cvpzf backup.tgz --exclude=/proc --exclude=/lost+found --exclude=/backup.tgz
--exclude=/mnt --exclude=/sys /
```

Now, lets explain this a little bit.

The 'tar' part is, obviously, the program we're going to use.

'cvpzf' are the options we give to tar, like 'create archive' (obviously), 'preserve permissions' (to keep the same permissions on everything the same), and 'gzip' to keep the size down.

Next, the name the archive is going to get. backup.tgz in our example.

Next comes the root of the directory we want to backup. Since we want to backup everything; /

Now come the directories we want to exclude. We don't want to backup everything since some dirs aren't very useful to include. Also make sure you don't include the file itself, or else you'll get weird results.

You might also not want to include the /mnt folder if you have other partitions mounted there or you'll

end up backing those up too. Also make sure you don't have anything mounted in /media (i.e. don't have any cd's or removable media mounted). Either that or exclude /media.

EDIT : kvidell suggests below we also exclude the /dev directory. I have other evidence that says it is very unwise to do so though.

Well, if the command agrees with you, hit enter (or return, whatever) and sit back&relax. This might take a while.

Afterwards you'll have a file called backup.tar.gz in the root of your filesystem, which is probably pretty large. Now you can burn it to DVD or move it to another machine, whatever you like!

EDIT2:

At the end of the process you might get a message along the lines of 'tar: Error exit delayed from previous errors' or something, but in most cases you can just ignore that.

Alternatively, you can use Bzip2 to compress your backup. This means higher compression but lower speed. If compression is important to you, just substitute the 'z' in the command with 'j', and give the backup the right extension.

That would make the command look like this:

Code:

```
tar cvpjf backup.tar.bz2 --exclude=/proc --exclude=/lost+found
--exclude=/backup.tar.bz2 --exclude=/mnt --exclude=/sys /
```

2: Restoring

Warning: Please, for goodness sake, be careful here. If you don't understand what you are doing here you might end up overwriting stuff that is important to you, so please take care!

Well, we'll just continue with our example from the previous chapter; the file backup.tar.gz in the root of the partition.

Once again, make sure you are root and that you and the backup file are in the root of the filesystem.

One of the beautiful things of Linux is that This'll work even on a running system; no need to screw around with boot-cd's or anything. Of course, if you've rendered your system unbootable you might have no choice but to use a live-cd, but the results are the same. You can even remove every single file of a Linux system while it is running with one command. I'm not giving you that command though!

Well, back on-topic.

This is the command that I would use:

Code:

```
tar xvpfz backup.tar.gz -C /
```

Or if you used bz2;

Code:

```
tar xvpfj backup.tar.bz2 -C /
```

**WARNING: this will overwrite every single file on your partition with the one in the archive!**

Just hit enter/return/your brother/whatever and watch the fireworks. Again, this might take a while.

When it is done, you have a fully restored Ubuntu system! Just make sure that, before you do anything else, you re-create the directories you excluded:

Code:

**mkdir proc**  
**mkdir lost+found**  
**mkdir mnt**  
**mkdir sys**  
**etc...**

And when you reboot, everything should be the way it was when you made the backup!

## 2.1: GRUB restore

Now, if you want to move your system to a new harddisk or if you did something nasty to your GRUB (like, say, install Windows), You'll also need to reinstall GRUB.

There are several very good howto's on how to do that here on this forum, so i'm not going to reinvent the wheel. Instead, take a look here:

Variant #1:

Restore GRUB quite simple in Ubuntu, instead going through all the "gain root access" and play with shell commands, you can use the Ubuntu installation CD to restore it without going through all kinds of hassles.

Here are the steps:

1. Boot your computer up with Ubuntu CD
2. Go through all the process until you reach "[!!!] Disk Partition"
3. Select Manual Partition
4. Mount your appropriate linux partions

/  
/boot  
swap  
.....

5. DO NOT FORMAT THEM.
6. Finish the manual partition
7. Say "Yes" when it asks you to save the changes
8. It will give you errors saying that "the system couldn't install ....." after that
9. Ignore them, keep select "continue" until you get back to the Ubuntu installation menu
10. Jump to "Install Grub ...."
11. Once it is finished, just restart your computer

There are a couple of methods proposed. I personally recommend the second one, posted by remmelt, since that has always worked for me.

Variant#2:

1. Pop in the Live CD, boot from it until you reach the desktop.
2. Open a terminal window or switch to a tty.
3. Type "grub"
4. Type "root (hd0,6)", or whatever your harddisk + boot partition numbers are (my /boot is at /dev/sda7, which translates to hd0,6 for grub).
5. Type "setup (hd0)", or whatever your harddisk nr is.
6. Quit grub by typing "quit".
7. Reboot.

Well that's it! I hope it was helpful!

As always, any feedback is appreciated!

# Managing Mysql database (adding database and user)

```
#
# Connect to the local database server as user root
# You will be prompted for a password.
#
mysql -h localhost -u root -p
#
# Now we see the 'mysql>' prompt and we can run
# the following to create a new database for Paul.
#
mysql> create database pauldb;
Query OK, 1 row affected (0.00 sec)
#
# Now we create the user paul and give him full
# permissions on the new database
mysql> grant CREATE,INSERT,DELETE,UPDATE,SELECT on pauldb.* to paul@localhost;
Query OK, 0 rows affected (0.00 sec)
#
# Next we set a password for this new user
#
mysql> set password for paul = password('mysecretpassword');
Query OK, 0 rows affected (0.00 sec)
#
# Cleanup and ext
mysql> flush privileges;
mysql> exit;
#Removing database from Mysql
mysql> drop database database_name;
#Removing user from Mysql
mysql> drop user username;
```