Confirm Machine Architecture with arch

If you're looking to not just confiirm the architecture of the server you're logged into but also to use this knowledge in shell scripts, you can use the **arch command** instead of **uname**.

Just type **arch** in the command line, like this:

```
greys@s5:~ $ arch
x86_64
```

When I run this on one of my Raspberry Pi systems, I see a different output, because they are not x86 based processors:

```
greys@becky:~ $ arch
armv7l
```

See Also

- Unix Commands
- uname command
- Raspberry Pi OS

Unix Diff

diff is a mightly command line tool found in most of Unix and Unix-like operating systems. **diff** helps you to find differences between files and directories.

Things You Can Do with Unix Diff

- Compare files with diff
- Compare directories with diff
- Compare binary files with diff
- Compare backup copies to current files

How To Use Unix Diff

In its simplest form, compares two text files – you provide their names as command line options.

Let’s create two files first:

```
greys@maverick:~ $ touch try
greys@maverick:~ $ touch try2
```

Diff won’t show any difference because they’re exactly the same – empty new files:

```
greys@maverick:~ $ diff try try2
```

If we change one of the files by adding the hello word to it, see what happens:

```
greys@maverick:~ $ echo "hello" >> try2
greys@maverick:~ $ diff try try2
0a1
> hello
```

Diff spotted the difference and indicated, which file has it (> means second file, the file in the right section of the command line).

Now, let’s add something else to the first file to make things a bit more interesting:

```
greys@maverick:~ $ echo "hi" >> try
greys@maverick:~ $ diff try ttry2
1c1
< hi
---
> hello
```

See? diff now highlighted that the first file (< pointing to the file in the left part of the command line you specified) also has a line that’s different from second file.

If we now add exactly the same line to both files, diff will ignore it because it only shows what’s different:

```
greys@maverick:~ $ echo "test" >> try
greys@maverick:~ $ echo "test" >> try2
greys@maverick:~ $ cat try
hi
test
greys@maverick:~ $ cat try2
hello
test
greys@maverick:~ $
greys@maverick:~ $ diff try try2
1c1
```

```
< hi
---
> hello
```

That’s it for today! I’ll show you some advanced usages of the **diff command** some other time.

See Also

- Unix commands
- Comparing text files with diff
- Basic Unix commands

Linux: List All Users

Another very common request, both form my Unix/Linux beginner users and from the visitors to Unix Tutorial blog. Usually, user list is needed because you plan on doing something with it – so please leave a comment and let me know what it is. Who knows, there might be a quicker and easier way of doing the same!

List all users with getent

This is probably the quicked and easiest way of getting the list of users in your Linux system, along with most relevant info about each of them:

```
greys@ec2 ~]$ getent passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

```
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
systemd-bus-proxy:x:999:997:systemd Bus Proxy:/:/sbin/nologin
systemd-network:x:998:996:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:997:995:User for polkitd:/:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
chrony:x:996:993::/var/lib/chrony:/sbin/nologin
centos:x:1000:1000:Cloud User:/home/centos:/bin/bash
```

IMPORTANT: if your Linux system is part of an AD or LDAP infrastructure, the getent passwd command will get you all the users in AD, rather than just those locally created on your Linux server.

List all users from /etc/passwd

You can also just look at the contents of the **/etc/passwd** file: it will look very similar to the getent output:

```
[greys@ec-ws1 ~]$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

```
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/sbin/nologin
systemd-bus-proxy:x:999:997:systemd Bus Proxy:/:/sbin/nologin
systemd-network:x:998:996:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:997:995:User for polkitd:/:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
tss:x:59:59:Account used by the trousers package to sandbox the tcsd daemon:/dev/null:/sbin/nologin
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin
nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
chrony:x:996:993::/var/lib/chrony:/sbin/nologin
centos:x:1000:1000:Cloud User:/home/centos:/bin/bash
```

Extract usernames from passwd with awk

All these lists are fine, but they’re not easily actionable in scripts or any other command line processing in Unix. The reason for this is, of course, because we’re getting too much information: instead of just the list of usernames, we’re looking at lots of passwd fileds like full name, user id, group if, user shell and so on.

So the next step is probably extracting usernames from the output we received. Here’s how we can do it: we’ll use the awk field separator to split fields.

Here’s the result:

```
[greys@ec-ws1 ~]$ cat /etc/passwd | awk -F: '{print $1}'
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
operator
games
ftp
nobody
avahi-autoipd
systemd-bus-proxy
systemd-network
dbus
polkitd
rpc
tss
rpcuser
nfsnobody
postfix
sshd
chrony
centos
```

That’s it for today! Stay tuned for more!

See Also

- User management commands
- How To: get username from UID

- getent command
- awk delimiter

---

apropos and whatis commands

As you know, each Unix/Linux distro comes with a massive set of man pages – helpful manuals for using pretty much every standard command found in your OS.

Quite often you don't know the command though, but know what it should do. In such a scenario, **apropos and whatis commands** may come in handy.

Man pages database

There's a special database (index) of man pages on your system, which indexes man pages and keeps a list of short 1-line descriptions of each command documented in a man page. This is how such a database looks in text form on a Ubuntu 16.04 Linux system:

```
...
ldap.conf (5) - LDAP configuration file/environment variables
adduser.conf (5) - configuration file for adduser(8) and addgroup(8) .
mailcap.order (5) - the mailcap ordering specifications
interfaces (5) - network interface configuration for ifup and ifdown
Compose (5) - X client mappings for multi-key input sequences
PAM (7) - Pluggable Authentication Modules for Linux
[ (1) - check file types and compare values
access.conf (5) - the login access control table file
accessdb (8) - dumps the content of a man-db database in a human readable format
```

```
add-apt-repository (1) - Adds a repository into the /etc/apt/sources.list or /etc/apt/sources.lis...
add-shell (8) - add shells to the list of valid login shells
addgroup (8) - add a user or group to the system
addpart (8) - tell the kernel about the existence of a partition
adduser (8) - add a user or group to the system
agetty (8) - alternative Linux getty
apropos (1) - search the manual page names and descriptions
apt (8) - command-line interface
...
```

Left side of the output lists command names, right side of the outout shows a brief command description. **apropos and whatis** commands work with these fields and allow you to search them.

Using whatis command

This command is useful when you want to confirm what a particular Unix command does. It searches man pages, but focuses specifically on the command names, rather than their descriptions.

For instance, if I know ls command, I would use whatis like this to find out more:

```
root@vps1:~# whatis ls
ls (1) - list directory contents
```

if I want to find similar commands, I can use the wildcard (in this example: all commands starting with **ls** combination):

```
root@vps1:~# whatis -w 'ls*'
ls (1) - list directory contents
lsattr (1) - list file attributes on a Linux second extended file system
lsb_release (1) - print distribution-specific information
lsblk (8) - list block devices
lscpu (1) - display information about the CPU architecture
lsinitramfs (8) - list content of an initramfs image
lsipc (1) - show information on IPC facilities currently
```

```
employed in the system
lslocks (8) - list local system locks
lslogins (1) - display information about known users in the
system
lsmod (8) - Show the status of modules in the Linux Kernel
lsof (8) - list open files
lspgpot (1) - extracts the ownertrust values from PGP keyrings
and list them in GnuPG ow..
```

Using apropos command

apropos is useful when you don’t remember the command but may have a few keywords describing its functionality.

Using ls command from the previous examples, we can find it using “directory” keyword. Of coruse, searching for “directory” will find all the commands which have anything to do with directories, as shown below:

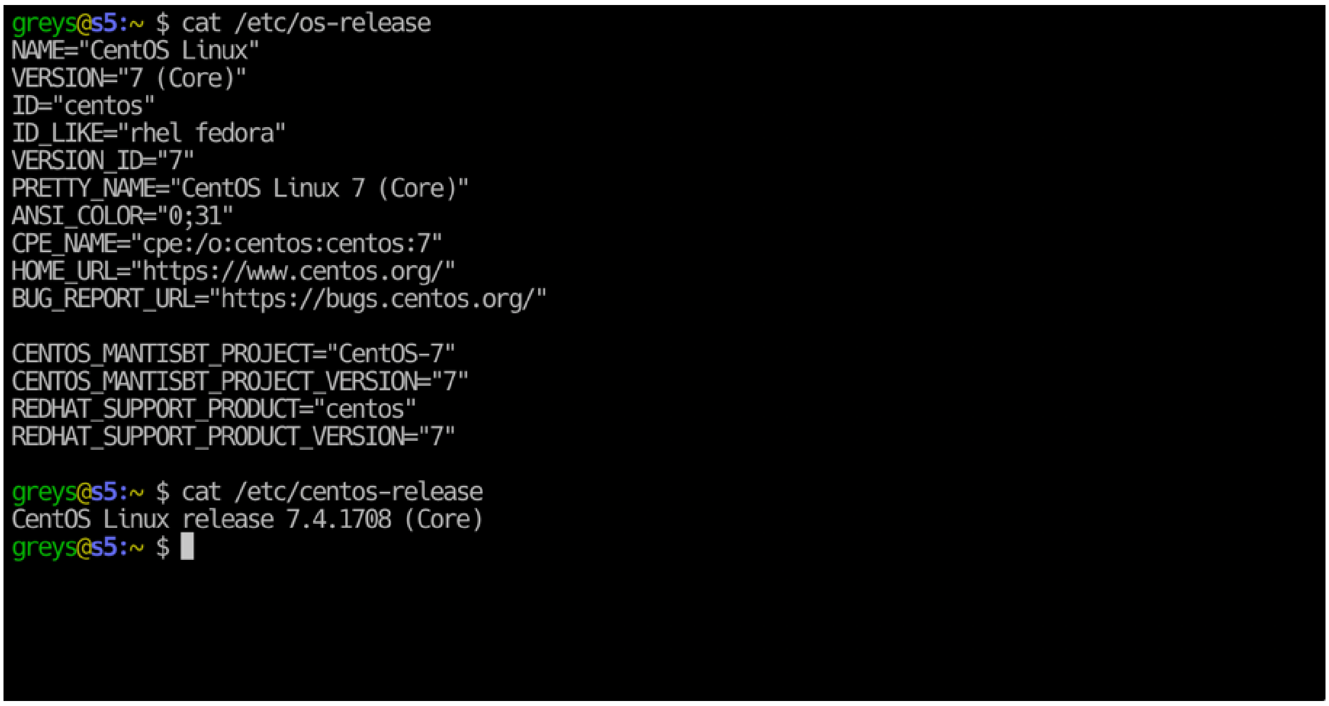
```
root@vps1:~# apropos directory
basename (1) - strip directory and suffix from filenames
bindtextdomain (3) - set directory containing message catalogs
chroot (8) - run command or interactive shell with special
root directory
dbus-cleanup-sockets (1) - clean up leftover sockets in a
directory
depmod.d (5) - Configuration directory for depmod
dir (1) - list directory contents
find (1) - search for files in a directory hierarchy
grub-macbless (8) - bless a mac file/directory
grub-mknetdir (1) - prepare a GRUB netboot directory.
helpztags (1) - generate the help tags file for directory
ls (1) - list directory contents
mklost+found (8) - create a lost+found directory on a mounted
Linux second extended file system
mktemp (1) - create a temporary file or directory
modprobe.d (5) - Configuration directory for modprobe
mountpoint (1) - see if a directory or file is a mountpoint
pam_mkhomedir (8) - PAM module to create users home directory
pwd (1) - print name of current/working directory
pwdx (1) - report current working directory of a process
```

```
run-parts (8) - run scripts or programs in a directory
vdir (1) - list directory contents
```

See Also

- Basic Unix commands
- Advanced Unix commands
- ls command
- What Is ln
- man command

How To: 5 Ways to Check CentOS Version



```
greys@s5:~ $ cat /etc/os-release
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"

greys@s5:~ $ cat /etc/centos-release
CentOS Linux release 7.4.1708 (Core)
greys@s5:~ $
```

One of the very first questions a Linux user asks is about confirming the release (OS version) in use. Knowing release helps with highlighting software dependencies and compatibilities, confirms availability of certain features in your OS and simplifies the process of system administration –

certain releases have a preferred set of commands for day-to-day management.

With CentOS being a rather popular server grade Linux distro, I can see that many visitors of my blog look for the same guidance quite regularly: **check CentOS version**. This article introduces 5 of the most common ways to do just that.

1. Inspect /etc/system-release

Just to be super sure that you're actually looking at a CentOS distribution of Linux, I suggest you start with the /etc/os-release file. As shown below, it will help you with confirming your Linux distro and its major release version (CentOS and 7 in my case):

```
greys@s5:~ $ cat /etc/os-release
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"
```

Now that we're sure it's CentOS, let's look into the /etc/centos-release file – this will show you the full release version of your operating system:

```
greys@s5:~ $ cat /etc/centos-release
CentOS Linux release 7.4.1708 (Core)
```

Interesting: if you’re coming from RedHat infrastructure, you’d normally be looking for **/etc/redhat-release** file. That’s okay and the good news is this will still work in CentOS:

```
greys@s5:~ $ cat /etc/redhat-release
CentOS Linux release 7.4.1708 (Core)
```

In fact, if you look at the **/etc/redhat-release** file on a CentOS server closely, you’ll notice that it is a symbolic link to **/etc/centos-release**:

```
greys@s5:~ $ ls -ald /etc/redhat-release
lrwxrwxrwx. 1 root root 14 Sep 18 2017 /etc/redhat-release ->
centos-release
```

2. Use hostnamectl to confirm Linux release

Provided that you’re running a recent enough version of Linux, you should have the hostnamectl command installed.

Among other things, hostnamectl provides easy access to the OS release information and Linux kernel version:

```
greys@s5:~ $ hostnamectl
Static hostname: s5.ts.im
Icon name: computer-desktop
Chassis: desktop
Machine ID: 5f7e36c18a974f06ae94ddaaf11d71e8
Boot ID: 337e48b00fed4abe9ab929fed5aa6018
Operating System: CentOS Linux 7 (Core)
CPE OS Name: cpe:/o:centos:centos:7
Kernel: Linux 3.10.0-693.11.6.el7.x86_64
Architecture: x86-64
```

3. Confirm CentOS version with rpm

Next option you have is to use the RPM package manager to query a special package named centos-release. It will include the exact CentOS release version right in its full package name:

```
greys@s5:~ $ rpm -qa centos-release
centos-release-7-4.1708.el7.centos.x86_64
```

if you're using RedHat, just do the same for the **redhat-release** package.

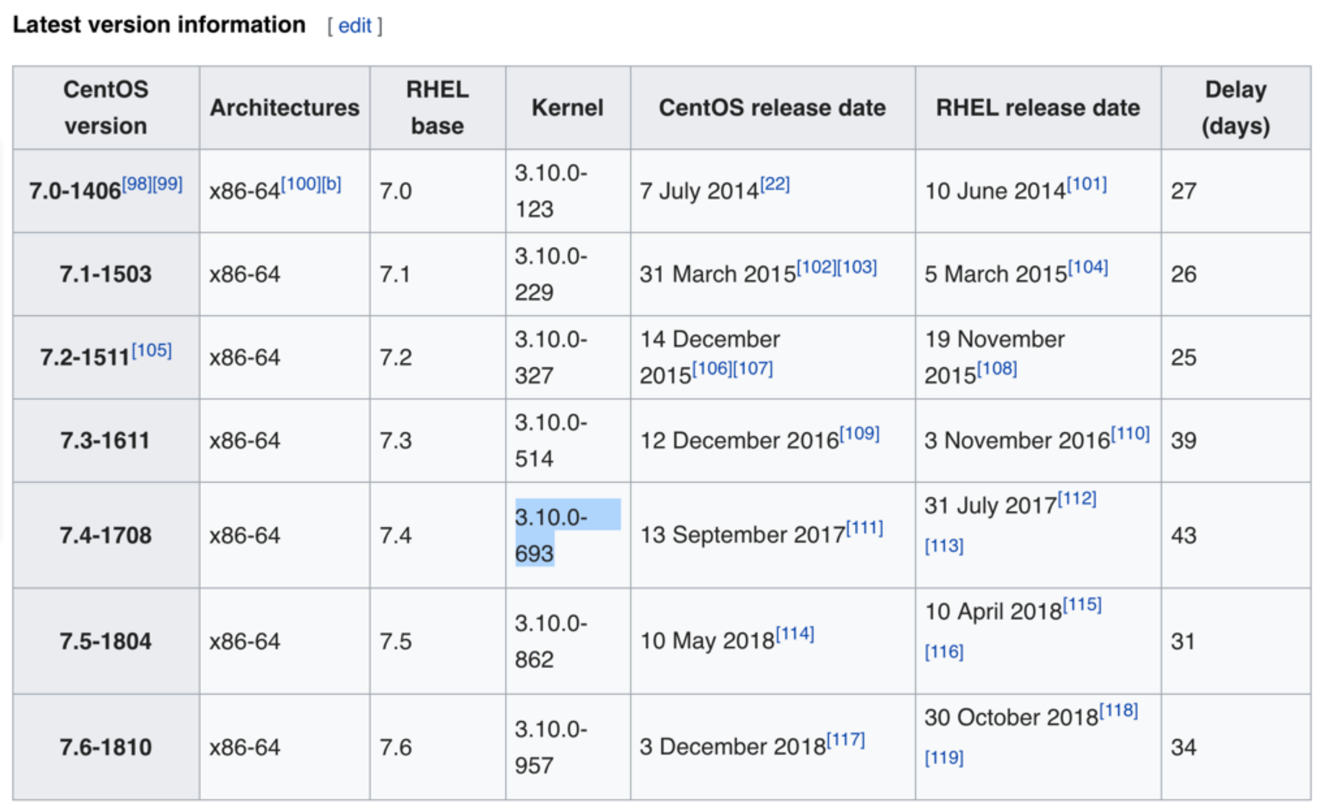
4. Confirm CentOS version using Linux kernel version

There are many ways to confirm your Linux kernel version, like uname command:

```
greys@s5:~ $ uname -r
3.10.0-693.11.6.el7.x86_64
```

Using the kernel version number – 3.10.0-693 in my example – you can confirm the CentOS release using one of the public version information pages, like the CentOS wikipedia page.

Once you browse to the CentOS wikipedia page, just search for the kernel version number and it should find something like this for you, confirming CentOS version to be **7.4-1708**:



Latest version information [edit]

CentOS version	Architectures	RHEL base	Kernel	CentOS release date	RHEL release date	Delay (days)
7.0-1406[98][99]	x86-64[100][b]	7.0	3.10.0-123	7 July 2014[22]	10 June 2014[101]	27
7.1-1503	x86-64	7.1	3.10.0-229	31 March 2015[102][103]	5 March 2015[104]	26
7.2-1511[105]	x86-64	7.2	3.10.0-327	14 December 2015[106][107]	19 November 2015[108]	25
7.3-1611	x86-64	7.3	3.10.0-514	12 December 2016[109]	3 November 2016[110]	39
7.4-1708	x86-64	7.4	3.10.0-693	13 September 2017[111]	31 July 2017[112][113]	43
7.5-1804	x86-64	7.5	3.10.0-862	10 May 2018[114]	10 April 2018[115][116]	31
7.6-1810	x86-64	7.6	3.10.0-957	3 December 2018[117]	30 October 2018[118][119]	34

5. Use lsb_release command to confirm Lunux release

Linux Standard Base (LSB) is a joint project by major Linux vendors to standardise configuration and usage of Linux distros. Amonth other things, it provides the lsb_release command that can help you check CentOS version.

Most likely, you'll have to install it first:

```
greys@s5:~ $ yum install redhat-lsb-core
...
```

Once installed, run the **lsb_release command** with the -d option:

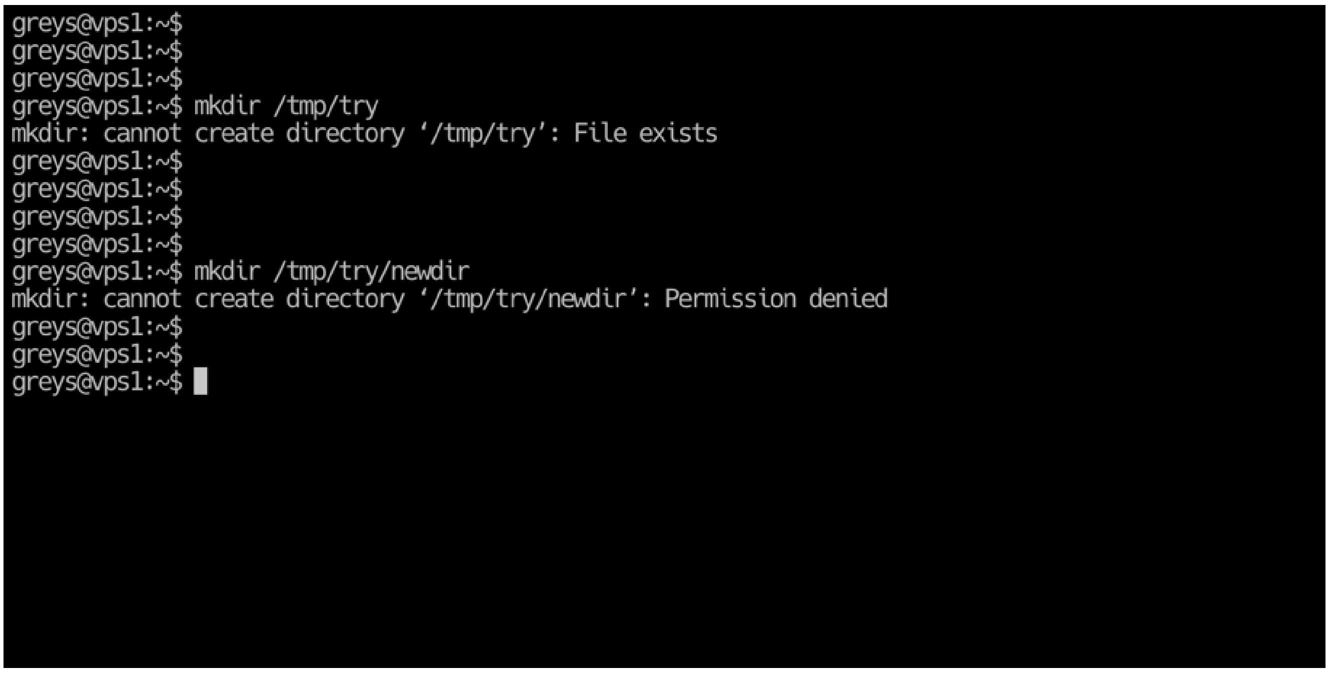
```
greys@s5:~ $ lsb_release -d
Description: CentOS Linux release 7.4.1708 (Core)
```

See Also

- Check Raspbian version
- Use /proc/version to identify your Linux release
- Find out Linux version using LSB
- uname command

---

mkdir cannot create directory



```
greys@vps1:~$
greys@vps1:~$
greys@vps1:~$
greys@vps1:~$ mkdir /tmp/try
mkdir: cannot create directory '/tmp/try': File exists
greys@vps1:~$
greys@vps1:~$
greys@vps1:~$
greys@vps1:~$
greys@vps1:~$ mkdir /tmp/try/newdir
mkdir: cannot create directory '/tmp/try/newdir': Permission denied
greys@vps1:~$
greys@vps1:~$
greys@vps1:~$
```

New Linux users often get puzzled by the "**mkdir: cannot create directory**" errors when taking first steps and trying to learn basics of working with files and directories. In this short post I'll show the two most common types of this mkdir error and also explain how to fix things so that you no longer get these errors.

mkdir: cannot create directory – File exists

This should be self explanatory after a few weeks of using commands like **mkdir**, but the first time you see this it can be confusing.

File exists? How can it be when you're just trying to create a directory? And why does it say "File exists" when you're trying to create a directory, not a file?

This error suggests that the directory name you're using (/tmp/try in my example shown on the screenshot) is already taken – there is a file or a directory with the same name, so another one can't be created. You can use the wonderful ls command to check what's going on:

```
greys@vps1:~$ ls -ald /tmp/try
drwxr-xr-x 2 greys root 4096 Nov 5 18:55 /tmp/try
```

Sure enough, we have a directory called /tmp/try already!

The reason it says "File exists" is because pretty much everything in Unix is a file. Even a directory!

Possible solutions to mkdir: cannot create directory – file exists scenario

Rename (move) existing directory

Use the mv command to move /tmp/try into some new location (or giving it new name). Here's how to rename /tmp/try into /tmp/oldtry:

```
greys@vps1:~$ mv /tmp/try /tmp/oldtry
```

Let's rerun the mkdir command now:

```
greys@vps1:~$ mkdir /tmp/try
```

…and since there are no errors this time, we probably have just created the /tmp/try directory, as desired. Let's check both /tmp/try and the /tmp/oldtry with ls:

```
greys@vps1:~$ ls -ald /tmp/try /tmp/oldtry
drwxr-xr-x 2 greys root 4096 Nov 5 18:55 /tmp/oldtry
drwxrwxr-x 2 greys greys 4096 Nov 5 19:08 /tmp/try
```

Remove existing file

Another option you always have is to simply remove the file that's blocking your mkdir command.

First, let's create an empty file called **/tmp/newtry** and confirm it's a file and not a directory usng ls command:

```
greys@vps1:~$ touch /tmp/newtry
greys@vps1:~$ ls -lad /tmp/newtry
-rw-rw-r-- 1 greys greys 0 Nov 5 20:50 /tmp/newtry
```

Now, if we try mkdir with the same name, it will fail:

```
greys@vps1:~$ mkdir /tmp/newtry
mkdir: cannot create directory '/tmp/newtry': File exists
```

So, to fix the issue, we remove the file and try mkdir again:

```
greys@vps1:~$ rm /tmp/newtry
greys@vps1:~$ mkdir /tmp/newtry
```

This time there were no errors, and **ls command** can show you that indeed you have a directory called **/tmp/newtry** now:

```
greys@vps1:~$ ls -lad /tmp/newtry
drwxrwxr-x 2 greys greys 4096 Nov 5 20:50 /tmp/newtry
```

mkdir: cannot create directory – Permission denied

This is another very common error when creating directories using mkdir command.

The reason for this error is that the user you're running the mkdir as, doesn't have permissions to create new directory in the location you specified.

You should use ls command on the higher level directory to confirm permissions.

Let's proceed with an example:

```
greys@vps1:/tmp$ mkdir try2018
greys@vps1:/tmp$ mkdir try2018/anotherone
greys@vps1:/tmp$ ls -ald try2018
drwxrwxr-x 3 greys greys 4096 Nov 5 21:04 try2018
```

All of these commands succeeded because I first created new directory called try2018, then another subdirectory inside of it. ls command confirmed that I have 775 permissions on the try2018 directory, meaning I have read, write and execture permissions.

Now, let's remove the write permissions for everyone for directory try2018:

```
greys@vps1:/tmp$ chmod a-w try2018
greys@vps1:/tmp$ ls -ald try2018
dr-xr-xr-x 3 greys greys 4096 Nov 5 21:04 try2018
```

If I try creating a subdirectory now, I will get the **mkdir: cannot create directory – permissions denied** error:

```
greys@vps1:/tmp$ mkdir try2018/yetanotherone
mkdir: cannot create directory 'try2018/yetanotherone':
Permission denied
```

To fix the issue, let's add write permissions again:

```
greys@vps1:/tmp$ chmod a+w try2018
greys@vps1:/tmp$ mkdir try2018/yetanotherone
```

As you can see, try2018/yetanotherone directory was successfully created:

```
greys@vps1:/tmp$ ls -ald try2018/yetanotherone
```

```
drwxrwxr-x 2 greys greys 4096 Nov 5 21:05
try2018/yetanotherone
```

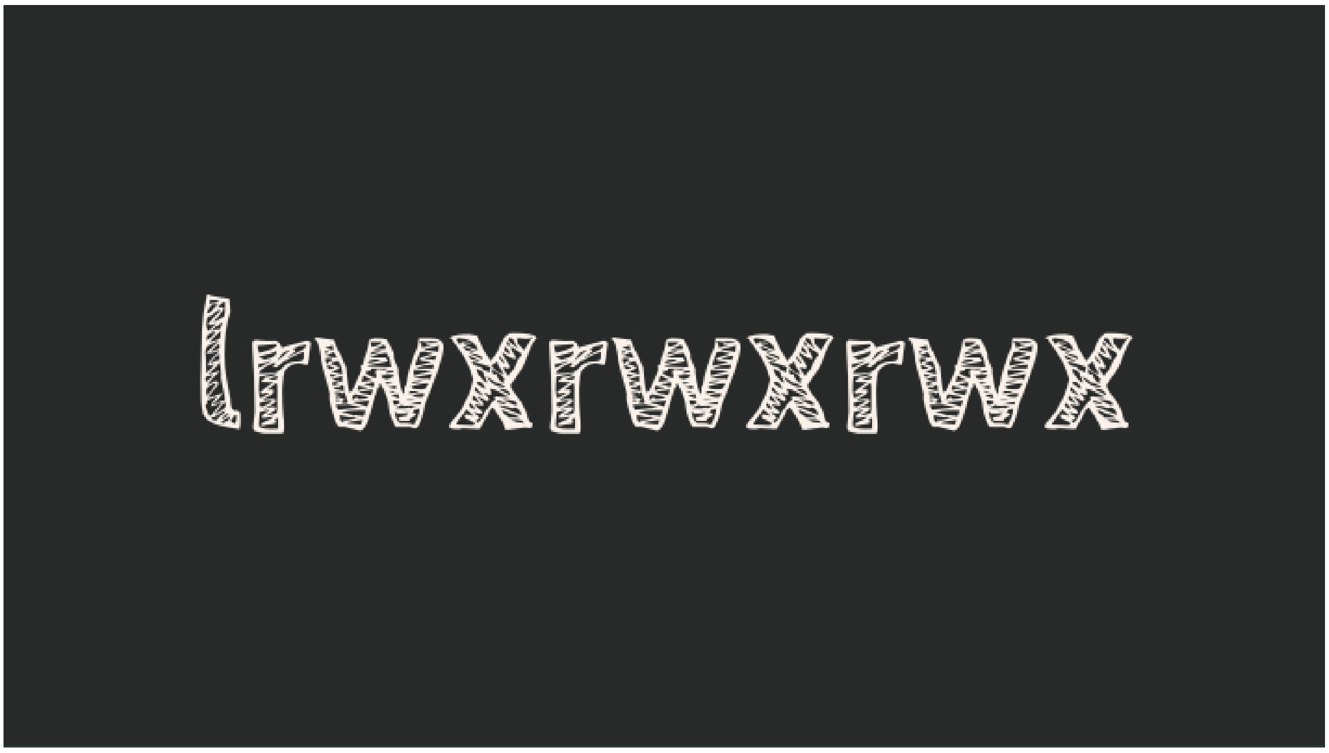
That's it for today! Hope you liked this tutorial, be sure to explore more basic Unix tutorials on my blog.

See Also

- Basic Unix commands
- mkdir command in Unix
- File types in Unix
- chmod and chown
- Unix commands tutorial

---

lrwxrwxrwx



lrwxrwxrwx permissions

If you come across a rather cryptic word “**lrwxrwxrwx**” when listing files and directories, here’s how you can interpret it.

As you know, file permissions in Unix are traditionally provided using 3 levels:

- **user (file owner) permissions** – specifically, permissions for the user currently setup as the file owner
- **group permissions** – since each file belongs to a particular group, this permission level confirms the access other group members will enjoy
- **other** (everyone else on the system) permissions

lrwxrwxrwx permissions

lrwxrwxrwx follows a permissions structure:

tUUUGGGOOO, where **t** is the file type indicator, **UUU** are the three characters specifying user (file owner) permissions, **GGG** are the group permissions and **OOO** are the others permissions.

So in the **lrwxrwxrwx** case, l stands for symbolic link – a special kind of pointer allowing you to have multiple filenames pointing to the same Unix file.

rwxrwxrwx is a repeated set of permissions, **rwx** meaning the maximum permissions allowable within basic settings.

Meaning of rwx

rwx permissions mean the following access is permitted:

- **r** – read
- **w** – write
- **x** – execute (or change directory)

Interestingly, lrwxrwxrwx is a permission that's rather uncommon: usually symlinks get a different (less forgiving) file permissions. Since symlinks are just pointers to other files, it doesn't matter much if you provide w (write) permissions or not – they would not allow you to control write access to the destination file.

Example: we use the touch command to create a simple file called **"file".** We specifically remove write permissions and this means we can't write anything into the file as you can see:

```
greys@maverick:~ $ touch file
greys@maverick:~ $ ls -al file
-rw-r--r-- 1 greys staff 0 3 Oct 23:36 file
greys@maverick:~ $ chmod u-w file
greys@maverick:~ $ ls -al file
-r--r--r-- 1 greys staff 0 3 Oct 23:36 file
greys@maverick:~ $ echo test > file
-bash: file: Permission denied
```

If we create a symlink **file2** pointing to **file**, it will actually show first group of permission block (user permissions) to be **rwx**, so it may see you have write access to the file it's pointing to:

```
greys@maverick:~ $ ln -s file file2
greys@maverick:~ $ ls -al file*
-r--r--r-- 1 greys staff 0 3 Oct 23:36 file
lrwxr-xr-x 1 greys staff 4 3 Oct 23:37 file2 -> file
```

But if we try to write the same word "**test**" into **file2** symlink, we'll still get an error cause it's pointing to the file which only has read permissions.

Finally, if we allow write permissions on the **file** again, we can write into **file2** symlink and it will work just fine this time:

```
greys@maverick:~ $ chmod u+w file
greys@maverick:~ $ ls -al file*
```

```
-rw-r--r-- 1 greys staff 0 3 Oct 23:36 file
lrwxr-xr-x 1 greys staff 4 3 Oct 23:37 file2 -> file
greys@maverick:~ $ echo test > file2
greys@maverick:~ $ cat file
test
greys@maverick:~ $ cat file2
test
```

See also

- **How to Use ln command for making links and symlinks**
- Advanced Unix Commands
- Unix Commands
- chmod vs chown

Centralized BASH history with timestamps

For every Unix user, there comes a point where shell history suddenly becomes very relevant. You learn to consult it, then start recovering the last command, then switch to searching past commands history to save precious time normaly taken typing.
Shortly after such a point in your life, you'll probably want to enhance your shell history in two very common ways:

1. Make sure every terminal window can update AND access your centralized shell history. So you run a command or two in one window, then type "history" anywhere else and see them two commands right there.
2. Provide meanigful timeline, this is done with timestamps. Very simple and powerful change helps you see exactly when each command occured.

Here’s how you achieve both of these massive improvents to your history in BASH. Just add this to **/etc/bashrc** on your Linux system:

```
export HISTTIMEFORMAT="%d.%m.%y %T "
export HISTCONTROL=ignoredups:erasedupsshopt -s histappend
export
PROMPT_COMMAND="${PROMPT_COMMAND:+$PROMPT_COMMAND$'\n'}history
-a; history -c; history -r;"
export HISTCONTROL=ignoreboth
```

How-To: Ubuntu – Enable SSH

Secure Shell (SSH) allows secure communication between networked computers for such purposes as logging in to a remote computer, running some commands remotely, and transferring files (with the scp command).

By default SSH is not enabled in Ubuntu. There is an **ssh command** installed, but it is only a client, and only allows you to login remotely into another computer, not to allow others to login into yours.

To enable ssh in Ubuntu that you need to **install the OpenSSH server**. To do that just use **apt-get**:

```
$ sudo apt-get install openssh-server
```

If you prefer you can also search for openssh server in the Ubuntu Software Center and install it that way.

Once it is installed you need to enable it in the OpenSSH Server configuration. To do this open and edit the **/etc/ssh/ssh_config** file with superuser privileges:

```
sudo nano /etc/ssh/ssh_config
```

The nano program is a terminal based text editor, but if you prefer a graphical editor you can open it in gedit:

```
$ sudo gedit /etc/ssh/ssh_config
```

In that configuration file look for the Port 22 line and uncomment it by removing the preceding hash sign **#**. That's all you need to edit to get the SSH server working, but if you wish you can review, enable, and edit other configuration options.

Once you're done save the file and restart SSH (which was started automatically when openssh-server was installed) for changes to take effect:

```
$ sudo service ssh restart
```

Your Ubuntu machine will now be able to accept SSH logins and communications through its IP address or host domain.

See also

- Unix Reference: SSH
- Change Passphrase for your SSH key
- Passwordless SSH with encrypted homedir in Ubuntu

How To Mount DMG Files from Command Line in Mac OS

DMG files are proprietary disk image files used for software distribution in **Mac OS**. Providiing both password protection and bzip2-like compression, these files are perfect packaging medium.

Usually DMG files are opened automatically when you click them in Finder. They appear as a folder with files, but actually Finder mounts each DMG file as a separate filesystem and then shows you its contents. If you're observant enough, you'll see that in the left side panell of Finder you have all the active DMG filesystems listed and ready to be ejected once you finish copying the files or installing new software.

Sometimes you may want to download and mount DMG file using Mac OS command line, and in this post I'll show you how to do it

Why would you want to mount DMG files manually?

I've been business traveling quite a bit lately which means I'm most of the time away from my home computer. Naturally, I have configured Remote Desktop access so that I can use my iPad to access my desktop whenever I need, but sometimes it takes forever to do some simple things just because of the graphics environment overhead.

If you're like me, you'll probably find Remote Desktop over 3G to be pretty boring, and will want to do as much as you can via command line.

Mounting DMG with hdiutil command

In order to manually mount DMG file, you'll need to use **hdiutil** command. You don't have to be a privileged user, so can do it as your own user.

For this example, I'm going to use the command line interface (CLI) for the excellent **HandBrake** tool, which is great for converting all sorts of videos into iPad and iPhone friendly resolution and mp4 format

Let's mount the image from my dmg file:

```
MacPro:~greys$ hdiutil attach /Users/greys/HandBrake-0.9.8-MacOSX.6_CLI_x86_64.dmg/dev/disk4
Apple_partition_scheme/dev/disk4s1
Apple_partition_map/dev/disk4s2          Apple_HFS
/Volumes/HandBrake-0.9.8-MacOSX.6_CLI_x86_64
```

As you can see from this output, the mount was successful and you now have the filesystem from DMG package available under the /Volumes/HandBrake-0.9.8-MacOSX.6_CLI_x86__64 directory.

Don't want to to take my word for it? Let's use the standard **mount** command to confirm that indeed we now have an new filesystem mounted:

```
MacPro:~ root# mount | grep HandBrake/dev/disk4s2 on
/Volumes/HandBrake-0.9.8-MacOSX.6_CLI_x86_64 (hfs, local,
nodev, nosuid, read-only, noowners)MacPro:~ root# cd
/Volumes/HandBrake-0.9.8-
MacOSX.6_CLI_x86_64/MacPro:HandBrake-0.9.8-MacOSX.6_CLI_x86_64
root# ls.Trashes HandBrakeCLI doc
```

Ejecting mounted DMG images from command line

Once you are done with whatever you were trying to do, there's no longer a reason to keep your DMG image mounted, so you should unmount it. While it’s possible to use umount command, I think it makes more sense if you use the same hdiutil tool that helped you mouunt the DMG image in the first place.

Here’s how you can eject the DMG image using hdiutil:

```
MacPro:~ greys$ hdiutil eject /Volumes/HandBrake-0.9.8-MacOSX.6_CLI_x86_64/"disk4" unmounted."disk4" ejected.
```

That’s it for today, hope you liked the post! Let me know!