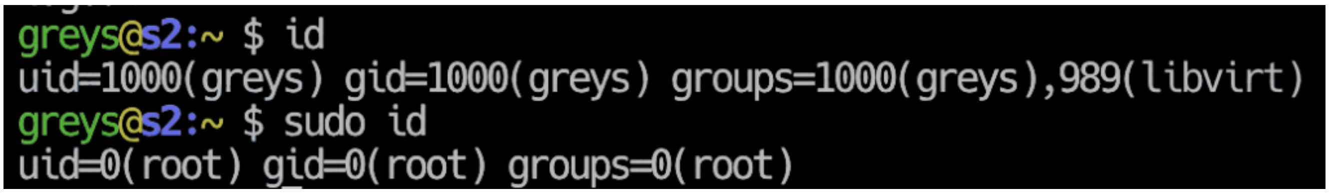
sudo Meaning: What Does sudo Mean?



```
greys@s2:~ $ id
uid=1000(greys) gid=1000(greys) groups=1000(greys),989(libvirt)
greys@s2:~ $ sudo id
uid=0(root) gid=0(root) groups=0(root)
```

sudo allows you to run a Unix command as a different user. Many beginner users are asking for meaning of the sudo command, so here's my take.

What sudo does

Using **/etc/sudoers** file to confirm what privileges are available to you, sudo command effectively elevates your access rights, thus allowing you to run commands and access files which would otherwise be not available to you. **sudo** runs these commands as root by default.

sudo meaning

The meaning of sudo command is:

- **su** (switch user)
- **do** action (run the specified command under specified user)

Default user in sudo

Unless specified, user is assumed to be **root**. So when you're running some sommand using **sudo**, your specified command is executed as **root**.

Using one of the most basics examples: **id command** shows your current username, its user id (UID), group id (GID) and group membership:

```
greys@s2:~ $ id
uid=1000(greys) gid=1000(greys)
groups=1000(greys),989(libvirt)
```

When we're running id using **sudo**, we're asking sudo to first become user root and to then run the specified command (id) as root:

```
greys@s2:~ $ sudo id
uid=0(root) gid=0(root) groups=0(root)
```

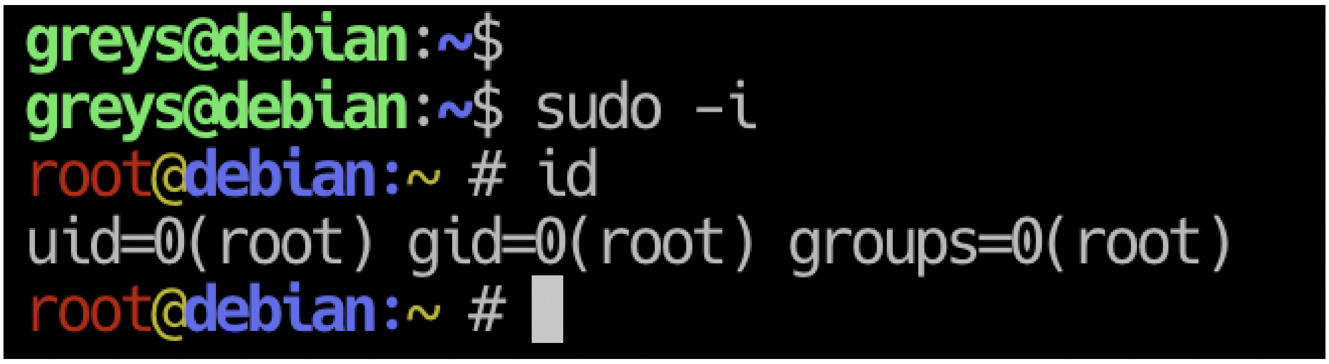
As you can see from the output, we get all the root information returned: UID =0, GID=0.

See Also

- **sudo reference**
- sudo command

- su command
- sudo tutorial
- visudo tutorial
- **Advanced Unix Commands**

How To: Setup sudo in Debian



```
greys@debian:~$
greys@debian:~$ sudo -i
root@debian:~ # id
uid=0(root) gid=0(root) groups=0(root)
root@debian:~ #
```

sudo in Debian Linux

Apparently, **Debian** installer doesn't install or activate **sudo** by default. This means that sudo command is not found the only privilege escalation method available is becoming root via su command. Since I like and use **sudo** daily, I decided to install and setup it on **Debian** VM.

Install sudo package in Debian

That's the very first step you'll need to do: use apt to install sudo. You need to become root before you do it, of course (so you must know root user password for your **Debian** install):

```
greys@debian:~$ su -
Password:
root@debian:~ # apt install sudo
Reading package lists… Done
Building dependency tree
Reading state information… Done
The following NEW packages will be installed:
  sudo
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 0 B/1,245 kB of archives.
After this operation, 3,886 kB of additional disk space will be used.
Selecting previously unselected package sudo.
(Reading database … 174742 files and directories currently installed.)
Preparing to unpack …/sudo_1.8.27-1_amd64.deb …
Unpacking sudo (1.8.27-1) …
Setting up sudo (1.8.27-1) …
Processing triggers for man-db (2.8.5-2) …
Processing triggers for systemd (241-5) …
root@debian:~ # sudo
usage: sudo -h | -K | -k | -V
usage: sudo -v [-AknS] [-g group] [-h host] [-p prompt] [-u user]
usage: sudo -l [-AknS] [-g group] [-h host] [-p prompt] [-U user] [-u user] [command]
usage: sudo [-AbEHknPS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p prompt] [-T timeout] [-u user] [VAR=value] [-i|-s] []
usage: sudo -e [-AknS] [-r role] [-t type] [-C num] [-g group] [-h host] [-p prompt] [-T timeout] [-u user] file …
```

Configure /etc/sudoers File

/etc/sudoers is the main configuration file for sudo command.

It contains list of users and groups that are allowed to become root (or become other users by invoking su command as root).

Here’s the default file in Debian 10 Buster:

```
root@debian:~ # cat /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/
instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
Defaults
secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
:/sbin:/bin"
# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include"
directives:
#includedir /etc/sudoers.d
```

I’ve highlighted the 3 most important elements of this file at this early stage:

root   ALL=(ALL:ALL) ALL

This is the line that allows you to debug sudo commands as root user.

At this means that any user that belongs to group sudo will also be allowed to use **sudo commands**:

%sudo   ALL=(ALL:ALL) ALL

Finally, this part includes additional configuration files from /etc/sudoers.d directory:

#includedir /etc/sudoers.d

… this means you don’t have to edit **/etc/sudoers** file but instead can create a specific file in **/etc/sudoers.d** and name it self-descriptively, like:

/etc/sudoers.d/web-server-admins

meaning, that this file will contain usernames and privileges required by web-server admins (usually commands like stopping/starting Apache or nginx webserver).

Since this is a very basic tutorial, we don’t have to edit the file at all – just need to add our user (mine is greys, as you remember) to the sudo group and check.

Add user to sudo group

Step 1: let’s make sure sudo is not accessible before we begin

This needs to be run as your regular user, not as root:

```
greys@debian:~$ sudo -i
[sudo] password for greys:
greys is not in the sudoers file.  This incident will be reported.
greys@debian:~$
```

Let’s check my groups just to be sure there’s no **sudo** among them:

```
greys@debian:~$ id greys
uid=1000(greys) gid=1000(greys) groups=1000(greys),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugdev),108(netdev),112(bluetooth),116(scanner)
```

Step 2: add user to sudo group

Excellent, now it’s time to add user greys to the group sudo

(we must become root again to run **usermod command**)

```
root@debian:~ # usermod -a -G sudo greys
root@debian:~ # id greys
uid=1000(greys) gid=1000(greys) groups=1000(greys),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),108(netdev),112(bluetooth),116(scanner)
```

As you can see, I’m now a member of the **sudo** group!

Step 3: Log out and log back in for group membership to be recognised

Now you need to disconnect from your server or desktop session and log in again, so that your group membersip is recognised. One reconnected, check your groups with **id command** and try **sudo** again:

```
greys@debian9:~$ id
uid=1000(greys) gid=1000(greys) groups=1000(greys),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),108(netdev),112(bluetooth),116(scanner)
```

so yes, we’re a member of sudo group now… This is the moment of truth! Let’s try to become root:

```
greys@debian:~$ sudo -i
root@debian:~ # id
uid=0(root) gid=0(root) groups=0(root)
```

Tha’ts it for today!

See Also

- **SUDO reference**
- **sudo command**
- Debian 10 Buster
- **How To Use visudo**
- sudo tutorial
- visudo tutorial

Ansible 2.0

If you’re managing configuration with Puppet or Chef, chances are you’ve heard of Ansible as well.

Just last week we got Ansible 2.0 released which brings quite a few improvents on top of a massive refactoring.

I’m quite late starting with Ansible but very impressed with it so far: it’s a great way of quickly confirming remote server’s state with SSH and sudo AND a neat way of scripting configurations with Ansible playbooks.

I have written my first playbook two weeks ago and need to change them now so that they follow the updated syntax.

Are you guys using Ansible as well?

How to update grub boot loader config

GRUB bootloader starts up what’s necessary for your Linux or UNIX system to boot up. You can edit its settings, like various boot options and which operating systems to select from, by editing the the */boot/grub/grub.cfg* or */etc/grub.conf* depending on your system. Graphical programs are also available for this purpose. See our GRUB Boot Loader overview for more.

Once you’ve edited your configuration you’ll need to update grub to use it. This is very easily done by this single command:

```
$ sudo update-grub
```

Then once you reboot your new config should be active.

See Also

- Unix Glossary
- sudo command
- Unix How-To

Passwordless SSH with

encrypted homedir in Ubuntu

Quite recently I came across a very interesting issue: while configuring passwordless SSH (it's public key based, so depending on you have it configured it may not be completely passwordless) access to some of my VPS servers, I found that the same keypair just wouldn't work on one of the servers.

Not only that, but the behaviour was quite bizzare: upon my first attempt to connect the public key would get rejected and a regular password would be requested by the ssh session. But once I successfully logged in with my password, any subsequent ssh connections would happily authenticate by my public key and would let me in without a problem.

Those of you using home dir encrypiton in Ubuntu are probably smiling right now! □ But becase I have never consciously configured or used this feature, it took me a good few hours to troubleshoot the issue and come up with the fix.

Why public-key based SSH doesn't work with encrypted home directories

The answer is quite simple: before your server can decide whether you are providing a valid and trusted SSH key, it must read your public key stored in your homedir. But if your homedir is encrypted, this becomes a classical chicken-and-egg scenario – until you log in and therefore decrypt your homedir the server won't gain access to your public key. Only you wouldn't be needing the public key by then, would you?

Store your authorized SSH keys outside your encrypted home directory

If you happen to like your homedir encryption AND would like to use public/private key SSH authentication, there is a way out: you need to store your authorized keys outside of your encrypted homedir.

The usual access restrictions and directory/file permissions still apply, so the only thing you're changing is moving your authorized keys outside of the encrypted homedir on your server. This way things will work exactly as you expect: you authenticate with your private key and this results in your automatically mounted and decrypted homedir.

Here are the steps to make this happen. You're going to need superuser privileges for my scenario because it caters for all the users on your Ubuntu server, not just one account that belongs to you (use sudo to become root).

Step 1: create a directory structure for your authorized keys.

First, the main directory, I created it under **/var** – seems quite a safe choice since this directory is unlikely to grow and is equally unlikely to get removed by accident.

```
# mkdir /var/openssh
```

Perfect! Now we need to create user-specific directories, just to keep this dir really tidy. My username is "**greys**", so here is the directory:

```
# mkdir /var/openssh/greys
# chown greys /var/openssh/greys
```

Step 2: copy existing authorized keys file into new location

(you must log in as your username for this, otherwise the homedir will stay encrypted)

```
$ cp /home/greys/.ssh/authorized_keys /var/openssh/greys
```

Step 3: update SSHd config with new location for

authorized_keys file

You're going to do this as root once again:

```
# vi /etc/ssh/sshd_config
```

update the value of the AuthorizedKeysFile so that it looks like this:

```
AuthorizedKeysFile       /var/openssh/%u/authorized_keys
```

Step 4: Restart SSH service

```
# service ssh restart
ssh start/running, process 3708
```

That's it! Give it a try and let me know how it worked out.

Recommended books:

[AMAZONPRODUCTS asin="1590594762″]

See also

- ssh – secure shell
- **How To Enable Secure Shell service in Ubuntu**
- Passwordless SSH
- Changing passphrase to your SSH key