How To Find Out User ID in Unix

There’s quite a few ways to confirm a user ID (uid) in Unix.

id command

This is probably one of the easiest ways to find out a uid of a particular user in your system:

```
# id -u greys
500
```

The most common way of using the id command is even simpler, and it gives you all the information about a user you may need:

```
# id greys
uid=500(greys) gid=500(greys) groups=500(greys)
```

This not only shows you the user id (uid), but also confirms user’s group id (gid) and all the rest Unix groups a user belongs to.

getent

Another way to get information about a user (including uid) is to use the **getent command**:

```
# getent passwd greys
greys:x:500:500:Gleb Reys:/home/greys:/bin/bash
```

In this example, greys is my username, 500 is the uid, and the second 500 in this line is my Unix group id (gid).

/etc/passwd file

If you know that the user you’re looking at is a local user (it is created on the local Unix system of yours), you can grep for it in /etc/passwd file directly:

```
$ grep greys /etc/passwd
greys:x:500:500:Gleb Reys:/home/greys:/bin/bash
```

---

bash: Find Out the Version of Your Unix Shell

bash (Bourne Again SHell) comes with pretty much every Unix-like OS these days. If you ever wonder what exact version of **bash** shell you have on your system, here's how you find out: just use the –version parameter in the command line.

Using /bin/bash to tell its version

On RedHat Linux, this is how it might look:

```
bash-2.05b$ /bin/bash --version
GNU bash, version 2.05b.0(1)-release (i386-redhat-linux-gnu)
Copyright (C) 2002 Free Software Foundation, Inc.
```

On Solaris 10 (one of the latest OpenSolaris builds, actually), it's a very similar output as well:

```
-bash-3.2$ /bin/bash --version
GNU bash, version 3.2.25(1)-release (i386-pc-solaris2.11)
Copyright (C) 2005 Free Software Foundation, Inc.
```

If you don't know the location of bash binary

The first and most obvious thing is to use which command to confirm the possible location of your bash binary.

On Linux:

```
 bash-2.05b$ which bash
/bin/bash
```

In Solaris:

```
-bash-3.2$ which bash
/usr/bin/bash
```

As you can see, this command returns you the full path to a binary, bash in our case. Once you know the full binary name, run it like explained in the first part of this post to confirm the version. Bear in mind that **which** command does not always return a result.

If this didn't work, a more sophisticated way to confirm the version of bash on your RedHat Linux system is to use rpm:

```
bash-2.05b$ rpm -qa | grep bash
bash-2.05b-41.4
```

This commands queries the RPM database of your RedHat Linux and confirms which version of the bash RPM package is installed, which naturally matches the version of the bash itself.

Unix Sockets Tutorial

I've noticed how many people found other pages of this blog trying to find more information about Unix sockets, and so I thought it's about time we shed some light on this seeming mysterious, but really simple concept.

What is a Unix socket?

A Unix socket (the technically correct name for it is Unix domain socket, **UDS**) is a way of inter-process communication

(**IPC**) in Unix. Like almost everything in Unix, a socket is a file. It’s a special file, to be precise. Unix processes which want to communicate between each other use special set of functions to access the special file of a Unix socket, and easily exchange data in both directions.

In very simple terms, a **Unix socket is nothing but a byte stream** – a data transfer between processes running locally or on networked Unix systems.

Examples of Unix sockets

Most well know examples of Unix sockets are probably those servicing the graphics system on your Unix box: X11 server socket and some optional sockets of programs managing it, like GDM (Gnome Desktop Manager).

GDM socket

```
bash-2.05b$ ls -al /tmp/.gdm_socket
srw-rw-rw-    1 root     root            0 Oct 25 17:49
/tmp/.gdm_socket
bash-2.05b$ file /tmp/.gdm_socket
/tmp/.gdm_socket: socket
```

syslog socket

The message logging daemon, syslogd, uses /dev/log on most systems to accept new messages to be logged in log files. Here’s how this file looks in RedHat:

```
bash-2.05b$ ls -al /dev/log
srw-rw-rw-    1 root     root            0 Oct 25 17:49
/dev/log
bash-2.05b$ file /dev/log
/dev/log: socket
```

Types of sockets in Unix

Here is another thing which can be quite confusing about sockets in Unix – the classification.

On the highest level, there are two types of sockets: **Unix domain sockets** for IPC (**AF_UNIX**) and **Unix network sockets** using Internet family of protocols (**AF_INET**), most commonly referred to as Unix Internet sockets.

These two types essentially define a set of communication protocols supported by a socket. Unix domain sockets are for IPC (interprocess communication) only, which means they can only be used for processes running locally on the same Unix system and communicating to each other. The Internet sockets support protocols which allow you to connect processes between different Unix systems: IP protocol for the network communication, and TCP/UDP protocols for the transport.

Connection-oriented and connectionless sockets

Both Unix domain sockets and Unix Internet sockets can use reliable (guaranteed) and unreliable (best effort) approaches for establishing and maintaining connections. With Unix domain sockets it doesn't really matter, as all the communication is local to your Unix system, but with Internet sockets it's a different story.

Connection-oriented sockets are called **stream sockets** (**SOCK_STREAM**), and are the reliable and guaranteed way to communicate. For Internet sockets, the **TCP** protocol is used to ensure that your data is confirmed to be delivered to the destination point, and your data packets will be received in the same order they were sent out.

The reason such sockets are called stream sockets is because your Unix will create and maintain a stream – an active connection between the source and the destination points, using TCP to manage this connection. All packets are automatically acknowledged and synchronization is maintained to ensure the ordered way of sending and receiving packets.

Connectionless sockets are called **datagram sockets** (**SOCK_DGRAM**) and they use **UDP** for their communication, which means that the packets of data which you send may or may not be received on the other end. Because there is no control over the transfer (no acknowledgments of the delivery, no ordered packet arrangements), there is no need to maintain such a connection. Hence the name – connectionless. You simply send a message out, and it's then a problem of a higher level protocol to ensure the data is transferred successfully.

See also

- Unix file types
- Unix Glossary

That's it – should be enough for a brief Unix sockets introduction. Do ask your questions in the comments to this post, and I'll be sure to answer them in future posts on Unix sockets – there's still plenty to show and explain!

---

sudo tutorial

sudo allows you to run a Unix command as a different user. Using **/etc/sudoers** file to confirm what privileges are available to you, this command effectively elevates your access rights, thus allowing you to run commands and access files which would otherwise be not available to you.

How sudo command works

The real and effective user id (uid) and group id (gid) are set to match those of the target user as specified in **/etc/sudoers** file (the safest way to change this file is to use the **visudo command** – check out the visudo tutorial). The way you use **sudo** is simple enough: you run this command and specify a command line you'd like to run with the privileges of a different user. Before the requested command is run, you are asked to confirm your identify by providing your user password.

id command, which shows you who you are in your Unix system (user id, user name, group id and other Unix groups you're member of), is the easiest way to demonstrate how your privileges are elevated. Truly, you become a different Unix user:

```
$ id
uid=1000(greys)   gid=33(www-data)   groups=33(www-data),113(admin)
$ sudo id
Password:
uid=0(root) gid=0(root) groups=0(root)
```

Using sudo in interactive mode

Sometimes you'll want to run many commands as a different user. Most common scenario for this presents in default sudo installation: you're encouraged to never become root, but instead use sudo to run commands as root.

When you want to use sudo in interactive mode, you run sudo with the -i parameter. This parameter causes sudo to imitate the initial login sequence – as if you simply log into your Unix system under a different user. The shell of this user is executed, and then you get the command prompt – anything you run after this will be run as the user granted to you by sudo:

```
$ id
uid=1000(greys)   gid=33(www-data)   groups=33(www-data),113(admin)
$ sudo -i
Password:
# id
uid=0(root) gid=0(root) groups=0(root)
```

```
# groups
root
```

Editing files with sudo

If instead of running some command as a different user you just want to edit some file which belogs to this user, then you will like the -e option for sudo. When using sudo to edit files, instead of a Unix command you specify the file you'd like to edit:

```
$  sudo -e /etc/passwd
```

Solaris Devices

This is a very brief introduction into navigating the device paths in Solaris. I'm using a Solaris 10 installed on Sun v490 for all the commands shown below.

Device files in Solaris

Even though all the block and character special device files are traditionally found under **/dev** directory, if you look closer at your Solaris 10 setup you will notice that they're not the device files themselves, but instead are just symbolic links to device files under **/devices** directory.

Solaris uses **/devices** directory for representing all the physical hierarchy of installed devices and buses found on your hardware system.

The directory tree under **/devices** copies the actual physical configuration of your hardware, and looks like this:

```
bash-3.00# ls /devices
```

```
iscsi                                   pci@8,700000:reg
iscsi:devctl                            pci@9,600000
memory-controller@0,400000              pci@9,600000:devctl
memory-controller@0,400000:mc-us3       pci@9,600000:intr
memory-controller@2,400000              pci@9,600000:reg
memory-controller@2,400000:mc-us3       pci@9,700000
options                                 pci@9,700000:devctl
pci@8,600000                            pci@9,700000:intr
pci@8,600000:devctl                     pci@9,700000:reg
pci@8,600000:intr                       pseudo
pci@8,600000:reg                        pseudo:devctl
pci@8,700000                            scsi_vhci
pci@8,700000:devctl                     scsi_vhci:devctl
pci@8,700000:intr
```

Most of these names are directories, so if you cd into **/devices/pseudo** directory, you will see all the software (hence pseudo) devices present in your system. Other directories will refer to various elements found on the system bus, for instance you can find a directory **/devices/pci@9,600000/SUNW,qlc@2/** which will represent a built-in FC-AL Host Adapter which manages hard drives on my system.

Device instance numbers

As you know, it’s quite possible to have more than one device of the same kind in your system. Because of this, all the physical devices are mapped to instance numbers. Even if you have only one device of a particular kind, it will get an instance number. Numeration starts with 0.

For example, on Sun v490 there are 2 on-board gigabit network interfaces, and they’re referred to as ce0 and ce1. Similarly, all other devices are numbered and mapped.

All the physical device mappings to their instances are recorded in **/etc/path_to_inst** file. That’s the file used by Solaris to keep instance numbers persistent across reboots:

```
bash-3.00# more /etc/path_to_inst # #       Caution! This file
contains critical kernel state
#
"/pseudo" 0 "pseudo"
"/scsi_vhci" 0 "scsi_vhci"
"/options" 0 "options"
"/pci@8,700000" 0 "pcisch"
"/pci@8,700000/ide@6" 0 "uata"
"/pci@8,700000/ide@6/sd@0,0" 1 "sd"
"/pci@8,600000" 1 "pcisch"
```

Unix Tutorial update: Reference page

Looking at this website access logs, I see how many people share the same problems and look for the same solutions, but use vastly different search queries to get to my posts.

I've decided to make your life easier, and have just launched a **Unix Tutorial Reference** page, which is an index of pages based on your searches. Most pages will have a basic introduction to the topic and provide further pointers to the solution posts.

Find Compiler Version in Unix

Finding the compiler version in your Unix system should be the first step before you attempt to compile any package from its source codes. In fact, if you're familiar with the common compilation routine, the **configure** script which you run to generate the **Makefile** before compiling anything does exactly that – it finds out which compilers (if any) you have installed on your system, and confirms their versions and capabilities.

If you want to find the compiler version yourself, here's what you do:

1) Confirm which compiler you're looking at

Most likely, it will be gcc, but since GCC isn't a GNU C Compiler anymore, but a GNU Compiler Collection, it can stand for any programming language from this suite. Here they are, just so that you know (a binary of each compiler is shown in bold):

gcc – GNU project C and C++ compiler
g++ – GNU project C and C++ compiler
gcj – GNU Compiler for Java
g77 – GNU project Fortran 77 compiler
gnat – GNU Ada Translator

2) Find where your compiler is installed

You have a pretty good chance of finding gcc binaries right under /usr/bin directory on most modern systems. Some older Unix distros will not have GCC installed by default, others like recent versions of Solaris and OpenSolaris, will have gcc under a different location. In Solaris 10, gcc binaries are under /usr/sfw/bin directory.

If the **gcc** binary isn't found, consider looking for the file using the **find** command, or query software repository to confirm if it's installed or now.

3) Run the compiler to find out its version

Most compilers will give you their version if -v option is specified:

```
$ gcc -v
Using built-in specs.
Target: x86_64-linux-gnu
Configured with: ../src/configure -v --enable-languages=c,c++,fortran,objc,obj-c++,treelang --prefix=/usr --enable-shared --with-system-zlib --libexecdir=/usr/lib --without-included-gettext --enable-threads=posix --enable-nls --program-suffix=-4.1 --enable-__cxa_atexit --enable-clocale=gnu --enable-libstdcxx-debug --enable-mpfr --enable-checking=release x86_64-linux-gnu
Thread model: posix
gcc version 4.1.2 (Ubuntu 4.1.2-0ubuntu4)
```

See also:

- Find out the release version of your Unix
- GCC, the GNU Compiler Collection

---

tee: Replicate Standard Output

Now and then I come across a situation when I need to run a script or a Unix command and would like to not only see the output of it on the screen, but also save this output to some log file. Redirecting the standard output using standard Unix stream redirection isn't always useful because your output will either be shown to you, or sent to the file – but not both at the same time

tee command

That's where the tee command becomes really useful. You pipe your output to this command, and let it take care of the rest.

Here's how you use it:

```
greys@simplyunix:~$ uname -a | tee /tmp/uname.txt
Linux simplyunix.com 2.6.16.29-xen #1 SMP Sun Sep 30 04:00:13
UTC 2007 x86_64 GNU/Linux
greys@simplyunix:~$ cat /tmp/uname.txt
Linux simplyunix.com 2.6.16.29-xen #1 SMP Sun Sep 30 04:00:13
UTC 2007 x86_64 GNU/Linux
```

What happens is that **tee** replicates the standard output to both your session and the specified file.

See also:

- script: Save Your Session Log

How To Find Out RedHat Version And More

If you're a really curious mind, you won't be satisfied with simply knowing the current release of your RedHat Linux, that's why there's a few more commands you could use to satisfy your interest.

RedHat release

If you simply want to confirm whether you're using a RHEL4, RHEL5 or any of the previous RedHat Linux releases, this is the first place to look:

```
bash-3.1$ cat /etc/redhat-release
Red Hat Enterprise Linux Client release 5 (Tikanga)
```

RedHat kernel version and type

Next step is to find out the exact Linux kernel version on your system, and also confirm whether it's 64-bit or not:

```
bash-3.1$ uname -a
Linux rhserver123 2.6.18-8.el5 #1 SMP Fri Jan 26 14:15:14 EST
2007 x86_64 x86_64 x86_64 GNU/Linux
```

RedHat kernel build

For the most curious ones, here's the last command. Use it to confirm who and when compiled the RedHat kernel you're using, and what gcc compiler was used in the build process.

```
bash-3.1$ cat /proc/version
Linux version 2.6.18-8.el5 (brewbuilder@ls20-
bc1-14.build.redhat.com) (gcc version 4.1.1 20070105 (Red Hat
4.1.1-52)) #1 SMP Fri Jan 26 14:15:14 EST 2007
```

See also

- How To Find Out the Release Version of Your Unix
- Check Raspbian version

How To Take A Screenshot in Unix (xwd)

Quite often there's a need for you to take a screenshot of your Unix desktop, and as always there's a number of ways to do it. Today I’m going to cover the command line approach to taking screenshots.

Taking a Screenshot with xwd

Most modern Unix desktop systems come with Gnome desktop environment by default, and use Xorg as their default X11 server. This means you are likely to have the xwd tool in your OS, which allows you to take screenshots.

Furthermore, many Unix distros come with hundreds of command-line tools bundled with the default OS install. Imagemagick is one of such bundled toolkits, and you can use the convert tool which is part of it for converting the xwd-generated screenshot into any graphics format of your preference.

Here’s how you use these two commands together:

```
bash-3.0$ xwd -root | convert - /tmp/screenshot.png
```

In this line, xwd command is invoked to take a full screenshot of your desktop. You then pipe its output to the convert tool and specify where you want this output saved to.

See Also

- Show off your distro screenshots with screenFetch