How To Create User Accounts in Unix

If you want to quickly create a new user account in your Unix OS, it ca be done with just one line

Adding new user accounts in Unix

To create a basic Unix user account with default settings, you need to know only one thing: the username.

The reason I say it as one word is because *username* (quite often referred to as “login”) is not the actual name of the new person gaining access to your Unix system, but rather a single keyword uniquely identifying this user in your system. Most often, usernames are derived from real names of users – **jsmith, johns** or **smithj** for John Smith, just to give you a few examples.

The simplest way to add a new user to your system is to do run a command like this:

ubuntu# **useradd jsmith**

If you don’t get any errors thrown back at you, this means your command was executed successfully and you now have a new user. Use this command to verify:

```
ubuntu# finger jsmith
Login: jsmith                     Name:
Directory: /home/jsmith                 Shell: /bin/sh
Never logged in.
No mail.
No Plan.
```

If you attempt to create a user with existing username, you’ll obviously get an error:

ubuntu# **useradd jsmith**

useradd: user jsmith exists

Setting a password for the newly created user account

Once you have created new user, you'll most likely need to have a new password assigned to it. Here's how you do it:

```
ubuntu# passwd jsmith
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

As you can see from the example, you'll be asked to type the new password twice, and it will be assigned to the user only if both inputs match.

See also:

- **How to list groups a user belongs to**
- **Find files which belong to a specific user**
- **Unix user management commands**

New section on this blog: Unix Commands

I've just integrated a new section: Unix Commands. The long-term plan is to have a categorized list of commands with most common usage documented in my typical easy-to-follow examples.

So far, there's not much, but I will referer to this section a lot in my future posts and its pages will have more examples for a particular Unix command compared to the original Unix Tutorial post where such a command is mentioned.

Have a look for yourself:

- **Unix Commands** – the main index
- **Basic Unix commands** – like it says, they really are basic. If you can think of something else which should be part of it, let me know
- **Advanced Unix commands** – to Unix gurus they'll seem basic as well, but my only guidance so far was that commands there will be the ones you don't have to use on a daily basis. As I add more pages, this section will most likely become something like "Most common Unix commands" and a set of really advanced commands will be added.

Let me know what you all think, and if there are some immediate candidates for any of the section – let me know and I'll add them to my list!

See Also

- Unix books reviews
- Unix Reference

Confirm the Day of the Week Based on a Timestamp

I recently created a Unix Questions and Answers page, if you have a Unix question – feel free to ask it there using the submit form and I'll do my best to help you out.

Today's Unix question is this:

How can we write a shell script in unix to find the day of the week when date is given?

The solution for this is even simpler: there's no need for Unix scripting, all you need is to have GNU date command at your disposal. I've already shown you all the basic date/time calculations using this great tool, and that's just another way of using it.

How to find a Day of the week based on timestamp

All you need is to know the base date. Let's say I'm interested in October 16th, 2009. Here's how easy it is to confirm that day will be Friday:

```
ubuntu$ date -d "Oct 16 2009" "+%a"
Fri
```

That's it – enjoy!

See also:

- **time and date in Unix**
- **date calculations with GNU date command**

How To Update atime and mtime for a File in Unix

If you remember, all files and directories in Unix filesystems have three timestamps associated with them – atime, ctime and mtime. Since questions about modifying access time (atime) and modification time (mtime) are quite frequent in my website logs, I thought I'd explain how it is done.

How to view atime, ctime and mtime

Before we go any further, I'd like to remind you that using stat command is probably the easiest way to look at all the three timestamps associated with each file:

```
ubuntu$ stat ./try
  File: `./try'
  Size: 0               Blocks: 0          IO Block: 4096
regular empty file
Device: 801h/2049d      Inode: 655596      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   greys)   Gid: (
113/   admin)
Access: 2008-11-17 05:01:16.000000000 -0600
Modify: 2008-11-17 05:01:16.000000000 -0600
Change: 2008-11-17 05:01:16.000000000 -0600
```

Even though ls command can be used to view the same times, we will depend on the stat command for today's post simply because it shows all the times together – it's great for explanations.

Modifying atime and mtime

There's a very simple way to update either atime or mtime for a given file, or even both at the same time: you should use the **touch command**.

Here's how it can be used to update the atime:

```
ubuntu$ touch -at 0711171533 ./try
```

The **-a** in the command line parameters refers to atime, while **-t** and the following sequence are nothing but a timestamp we want assigned to the file. In my example, 0711171533 means this:

- 07 – year of 2007
- 11 – November
- 17 – 17th

- 1533 – time of the day, 15:33 Now, if we run stat command again, you can see how the access time field got updated:

```
ubuntu$ stat ./try
  File: `./try'
  Size: 0               Blocks: 0          IO Block: 4096
regular empty file
Device: 801h/2049d      Inode: 655596      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   greys)   Gid: (
113/   admin)
Access: 2007-11-17 15:33:00.000000000 -0600
Modify: 2008-11-17 05:01:16.000000000 -0600
Change: 2008-11-17 05:01:48.000000000 -0600
```

Similarly, we can set the mtime, in my particular example it's the future – a day exactly one year from now.

-m is the command line option to specify that mtime is our main focus:

```
ubuntu$ touch -mt 0911171533 ./try
ubuntu$ stat ./try
  File: `./try'
  Size: 0               Blocks: 0          IO Block: 4096
regular empty file
Device: 801h/2049d      Inode: 655596      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   greys)   Gid: (
113/   admin)
Access: 2007-11-17 15:33:00.000000000 -0600
Modify: 2009-11-17 15:33:00.000000000 -0600
Change: 2008-11-17 05:05:41.000000000 -0600
```

Changing atime and mtime to the current Unix time

It's probably useful to know that the default behavior of the

touch command is to update both access time and modification time of a file, changing them to the current time on your system. Here's what will happen if I run touch against the

same file we used in all the examples:

```
ubuntu$ touch ./try
ubuntu$ stat ./try
  File: `./try'
  Size: 0               Blocks: 0          IO Block: 4096
regular empty file
Device: 801h/2049d      Inode: 655596      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/   greys)   Gid: (
113/   admin)
Access: 2008-11-17 05:09:33.000000000 -0600
Modify: 2008-11-17 05:09:33.000000000 -0600
Change: 2008-11-17 05:09:33.000000000 -0600
```

As you can see, all three fields have been reset to the new (current time) value. That's it for today, I hope this solved another one of your Unix mysteries!

See also

- mtime
- atime, ctime and mtime in Unix
- find out file permissions using Perl script
- time and date in Unix scripts

Ubuntu 8.10 – Intrepid Ibex – is here!

Right on time, next release of **Ubuntu Linux** has arrived. Ubuntu 8.10, codenamed Intrepid Ibex, is officially out.

Ubuntu 8.10

If you're into desktop goodness of **Ubuntu**, you'll probably

enjoy the Lifehacker.com screenshot tour.

To learn more details, visit these pages:

- Ubuntu 8.10 – desktop edition
- Ubuntu 8.10 – server edition

Download Ubuntu 8.10

If you can't wait any longer – grab a copy of Ubuntu from one of the sources presented at the Ubuntu download page.

See also:

Here are a few more links you might enjoy:

- Ubuntu – official page
- What's new in Ubuntu 8.10 server edition
- Ubuntu 8.10 – features

Today Only: Grab Your 100% Free Copy of CrossOver Pro

CodeWeavers are giving their award-winning CrossOver software for free.

What is CrossOver?

CrossOver is a tweaked and polished, proprietary version of Wine – an implementation of Windows API for Unix. Simply put, it's a software which allows you to run Windows applications on your Unix system. Wine, especially since the 1.0 release, is quite a pleasant and reliable way of running Windows software on your Unix desktop, but CrossOver is known for

putting many more tweaks on top of the features implemented in Wine, mostly to make popular office packages (Microsoft Office) and games work even better in emulated environment.

CrossOver Pro For Free

For one day only, **October 28th 2008**, you can go to the CodeWeavers website to request your free registration key for the professional version of their CrossOver software suite – either for Linux or Mac. These are fully functional serial keys for the pro version, technical support included.

The trick is that *you need to have this key activated as soon as possible!* Originally, you were only given time until midnight, but it is now promised that you'll get another 48hours to activate your key, although *free registration keys will stop after 23:59 PM Central Standard Time.*

Wait no more – the main CodeWeavers website is already down due to traffic, but Free CrossOver Pro registration form is still available!

See also:

- CodeWeavers website
- CrossOver @ Wikipedia
- Wine HQ

Tracking the Progress of Rsync Transfers

In my first introductory **rsync** post, **How To Synchronize Directories with Rsync**, I've shown you the most basic approach

to syncing two directories up. Today, I'd like to show you another useful thing you can do with **rsync**.

Just to remind you all, **rsync** is a ***r****emote* ***sync****hronization* tool. This means that its primary use lies in the filed of synchronizing files and directories between remote Unix systems, but I feel that it's really important for you to understand the basics before moving on to more advanced stuff.

The beauty of **rsync** is that its syntax and command line options are exactly the same for local and remote directories synchronizations, so it's not like you'll have to learn it all from scratch when I show you the real world usage of **rsync** in my next post.

rsync experiments setup

Just use the following commands to re-create your rsync playground for this post:

```
ubuntu$ rm -rf /tmp/dir1 /tmp/dir2
ubuntu$ mkdir /tmp/dir1 /tmp/dir2
ubuntu$ cd /tmp
ubuntu$ echo "original file 1" > dir1/file1
ubuntu$ echo "original file 2" > dir1/file2
ubuntu$ echo "original file 3" > dir1/file3
ubuntu$ cp dir1/file1 dir2
```

It will probably be more convenient for you to just download the same commands in form of a script though, so here's the link: rsync-setup.sh

Tracking rsync progress

The larger the directories you're synchronizing with rsync, the longer it will take for the command to finish. Depending on the scale of your task, it can be minutes, long hours or even days before you get the synchronization complete. With this in mind, the importance of progress tracking with rsync

should be obvious to you.

Here's how you make **rsync** report its progress: just use the **–progress** command line option (in addition to the command line I introduced you to last time):

```
ubuntu$ rsync -avz --stats --progress /tmp/dir1/ /tmp/dir2
building file list ...
4 files to consider
file2
          16 100%    0.00kB/s    0:00:00 (xfer#1, to-check=1/4)
file3
          16 100%   15.62kB/s    0:00:00 (xfer#2, to-check=0/4)

Number of files: 4
Number of files transferred: 2
Total file size: 48 bytes
Total transferred file size: 32 bytes
Literal data: 32 bytes
Matched data: 0 bytes
File list size: 59
File list generation time: 0.001 seconds
File list transfer time: 0.000 seconds
Total bytes sent: 193
Total bytes received: 64

sent 193 bytes  received 64 bytes  514.00 bytes/sec
total size is 48  speedup is 0.19
```

Why you should track the progress of rsyncs

The key differences are that you first get a regularly updated message about building a file list. When you run into hundreds of thousands of files, this becomes quite useful:

```
building file list ...
4 files to consider
```

Another way to track progress is to see the transfer progress

for each of the involved files. For thousands tiny files it won't be all that impressive a feature, but if you're transferring huge files between remote systems, a dynamic transfer progress for each file will be exactly what you need:

```
file2
           16 100%    0.00kB/s    0:00:00 (xfer#1, to-check=1/4)
file3
           16 100%   15.62kB/s    0:00:00 (xfer#2, to-check=0/4)
```

That's it, hope you like this new option. I promise to tell you more some other time!

See also:

- **Synchronize Directories in Unix with rsync command**
- **Comparing directories in Unix**
- **Finding large files and directories**

How To Remove Files and Directories with Special Characters

Today is going to be a practical tip. If you're managing many Unix systems, sooner or later you come across files with special characters – they can't be deleted with **rm** command using standard approach and require a bit of trickery to be successfully removed.

Examples of files with special characters

Any language apart from English will probably have special characters in the alphabet, but for the purpose of today's exercise I'll give you more standard examples: files starting with dash (-) and hash (#) characters:

```
ubuntu$ ls -al
-rw-r--r-- 1 greys admin    0 Sep 25 05:50 #try
-rw-r--r-- 1 greys admin    0 Sep 25 05:48 -try
```

Now, if you try to access these files or remove them, you will get errors:

```
ubuntu$ cat -try
cat: invalid option -- r
Try `cat --help' for more information.
ubuntu$ rm -try
rm: invalid option -- t
Try `rm ./-try' to remove the file `-try'.
Try `rm --help' for more information.
```

These errors happen because commands treat file names as command line options because they start with dash (-).

With filenames starting with hash (#), you'll get a different kind of error: your Unix shell will treat the rest of a filename (and anything that might follow it) as a comment because hashes are used to do it in shell scripts. That's why your cat command will not show any error but will not finish until you Ctrl+C it:

```
ubuntu$ cat #try
```

… and if you try removing such a file, you'll get a complaint from the **rm** command about missing command line parameters – because of the hash (#) sign, **rm** command receives no text as a parameter:

```
ubuntu$ rm #try
rm: missing operand
```

Try `rm --help' for more information.

How to remove a file when filename starts with dash (-)

First I’m gonna show you how to make your Unix shell interpret any filename directly instead of trying to analyze it as a set of command line options.

To make command ignore the leading dash (-) in a filename, use the – command line option:

ubuntu$ **rm -- -try**

As you can see, our file is gone:

ubuntu$ **ls -al**
-rw-r--r-- 1 greys admin   0 Sep 25 05:50 #try

Using backslash to escape special characters in a filename

Another option we have is to use a backslash (\), which will make shell interpreter ignore the special functionality of a character which immediately follows it. To escape the hash (#) our second file has, we should therefore do the following:

ubuntu$ **rm \#try**

Interesting to know: bash shell has an auto-completion functionality built in. When you type a filename, just press Tab key to make it auto-complete the name for you. Speaking of special characters in particular, quite a few of them are recognized by auto-completion and get escaped automatically.

So, if you start typing:

ubuntu $ **rm #t**

… and then press Tab, bash will not only auto-complete the name, but escape the leading hash (#):

ubuntu $ **rm \#try**

There's a few more tricks you can use for escaping special characters, but they're worth a separate post, so stay tuned! Until then, enjoy getting rid of annoying files with special characters in filenames!

See also:

- **Using math expressions in Unix shell**
- **Updating shell variables in Unix**
- **Parsing text in Unix shell**

Unix Tutorial Digest – September 21st, 2008

Here are the posts I've read and enjoyed this week:
* sshpass – automating SSH logins without passwordless RSA/DSA keys
* Tux Training: Get rid of command line histories with chattr command
* Tux Training: Speeding up your ext3 filesystem If you have any useful articles you'd like to share – just leave links in the comments area. Enjoy!

Easy date calculations in

Unix scripts with GNU date

When I was writing a post about using date command to confirm date and time in your Unix scripts, I made a note in my future posts list to cover the date calculations – finding out the date of yesterday or tomorrow, and so on. Today I'll show you the simplest way to calculate this.

GNU date command advantage

GNU version of the **date** command, although supporting a common syntax, has one great option: it allows you to specify the desired date with a simple string before reporting it back. What this means is that by default this specified date is assumed to be "now", but you can use other keywords to shift the result of the date command and thus show what date it was yesterday or a week ago:

Here's a default **date** output for the current date and time:

```
ubuntu$ date
Fri Sep 19 08:06:41 CDT 2008
```

Now, the parameter for specifying desired date is **-d** or **–date**, and if you use it with the "now" or "today" parameter, you'll get similar output:

```
ubuntu$ date -d now
Fri Sep 19 08:06:44 CDT 2008
ubuntu$ date -d today
Fri Sep 19 08:06:50 CDT 2008
```

Showing tomorrow's date

Similarly, you can get tomorrow's date:

```
ubuntu$ date -d tomorrow
Sat Sep 20 08:02:12 CDT 2008
```

Obviously, if you feel like specifying a format for the date, you can do it:

```
ubuntu$ date -d tomorrow "+%b %d, %Y"
Sep 20, 2008
```

Find out yesterday's date

Again, there's no rocket science involved in showing yesterday's date neither:

```
ubuntu$ date -d yesterday "+%b %d, %Y"
Sep 18, 2008
```

Show a date few days away

If you're interested in a certain date a few days or even weeks away, here's how you can do it:

Example 1: a date 5 days ago

```
ubuntu$ date -d "-5 days"
Sun Sep 14 08:45:57 CDT 2008
```

Example 2: a day 2 weeks from now

```
ubuntu$ date -d "2 weeks"
Fri Oct  3 08:45:08 CDT 2008
```

Example 3: a day two weeks ago from now

```
ubuntu$ date -d "-2 weeks"
Fri Sep  5 08:45:11 CDT 2008
```

Extreme example: a day 50 years ago

If you're really curious about dates in Unix, you can even make GNU date go back a few years:

```
ubuntu$ date -d "-50 years"
Fri Sep 19 08:47:51 CDT 1958
```

That's it for today, hope you like this little discovery –

having mostly worked with Solaris systems most of my career, I didn’t know my Ubuntu had this functionality bonus. Really useful!

See also:

- **Time and date in Unix scripts**
- **Variables in Unix shell scripts**
- **Updating shell variables in Unix**