

Managing Linux User Accounts

HERDING ACCOUNTS

The steps for setting up new accounts in Linux are automated and often use GUI-based tools. Under the hood, a number of mechanisms give the new user an environment to match his or her needs. In this month's Admin Workshop we discuss techniques for setting up accounts.

BY MARC ANDRÉ SELIG

With a modern Linux distribution, you just need a few clicks to create a new user account. Figure 1 shows you an example from Suse. GUI-based front-ends of this kind rely on command line programs such as *adduser* or *useradd* to do the heavy lifting.

It is extremely important for administrators to know exactly what goes on when they create a user account. Automated features can be tailored to meet the requirements in your environment. When you create an account, a number of databases need the new user's details. Users will typically need a home directory and write privileges to match. Administrators don't just throw their users in at the deep end when it comes to program settings; instead, many settings are pre-configured. Of course, it is impossible to take every setting into account when creating new users, espe-

cially if the user will not be logging in until some time in the distant future.

Creating a User Account

When creating a user account, the first step is to store information for that user in the central database. Assuming you do not have a distributed system such as NIS, NIS+, NetInfo or LDAP, but are simply managing the users on a stand alone computer, the file will be called */etc/passwd*. This is a text file with an entry for each account, including the login name and user ID, the group ID of the primary group, the home directory, and the preferred shell. You can also add details such as the user's first name, family name, or telephone number.

Unix systems previously stored an encrypted password that was used to authenticate the user in */etc/passwd*. The problem with this approach was that */etc/passwd* needed to be globally

readable, that is, the permissions were set to 0700. And that meant that anyone could read the encrypted passwords.

Today's processors do not take long to brute force weak passwords [1], so any user with access to the system has the opportunity to escalate his or her privileges.

To improve security, most of today's Unix derivatives store passwords in the */etc/shadow* file. The file is owned by the *root* user and the *shadow* group and read privileges are typically restricted to root and possibly the members of the shadow group. This prevents other users from accessing the passwords. A typical entry in */etc/passwd* looks like this:

```
mas:x:1000:1000:Marc Andre S
Selig,.,:/home/mas:/bin/bash
```

Some Unix systems do not use the password file directly but generate a binary

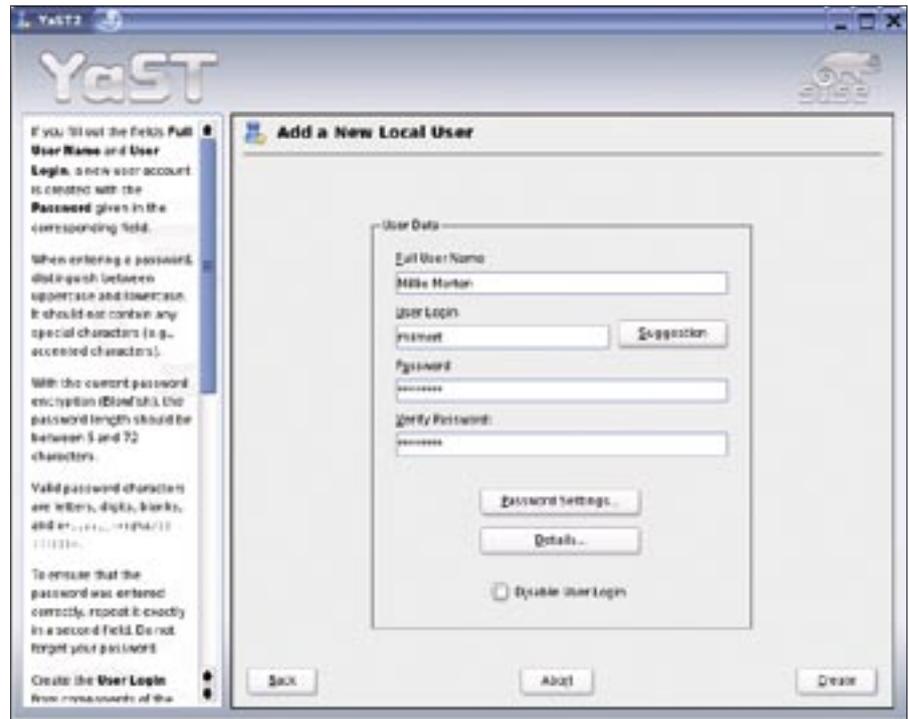


Figure 1: Creating user accounts on Suse is child's play. Scripts working in the background handle the details.

database from that file. BSD derivatives are notorious in this respect. To avoid conflicts and race conditions, there is *vipw*, which allows the root user to edit the password file. The tool runs the *vi* editor against */etc/passwd*, locking the file at the same time to prevent access by any other administrative users, and cleans up the binary user databases after the modifications have been completed.

Groups

User groups are a central feature of privilege management on any Unix system. Admins can create groups in a similar way to creating user accounts by running the *addgroup* or *groupadd* tools, for example. Groups are stored in a file called */etc/group*.

Each file created on a Unix machine is automatically assigned a user and a group. Admins can assign permissions separately for the user of the file and the

group. The ability to assign multiple users to a group makes it easy to grant these users access to specific files or directories.

Some typical examples of groups are *modem* or *dialout* for access to a modem port, *audio* for the sound card, and *cdrom* for assigning permissions for the CD-ROM drive. A database system could assign users permitted to access the database to a group called *db*, for example.

Users can belong to any number of groups, but each user has a primary group which specifies the group ownership for any files that user creates. The primary group is listed in */etc/passwd*, and any other memberships are listed in */etc/group*. The *id* command displays the group memberships for a user:

```
mas@ishi:~$ id
uid=500(mas) gid=100(users) 2
groups=100(users),14(uucp),
16(dialout),2
17(audio),33(video)
mas@ishi:~$
```

Modern Linux distributions have two different approaches to primary group assignments. Some set up a global *users* group for all human users of the system. Users can run the *chmod* command to

specify whether any files they create will be readable for the other group members (by setting the permissions to 0640: read and write permissions for the owner, read permissions for the group, deny permissions to all others), or not (0600: no permissions for the group or any other users).

Other distributions assign each user to a group of his or her own. This improves data protection slightly and allows users to set up granular teams by adding each other to their individual groups. This said, it makes more sense to set up specific task-related groups rather than just allowing things to develop.

There are a number of administrative groups for access control to programs or specific files. For example, the *wheel* group traditionally holds users who are allowed to run the *su* tool or equivalents. Today's distributions tend not to have a *wheel* group — just to keep things simple — and instead allow any user to run *su*. Groups such as *tty*, *disk*, and *lp* refer to system components (access to terminals, hard disks and printers); and the *bin* or *sys* groups allow administrators to specify which accounts can launch which programs. These groups are never assigned to human users.

The special *nobody* (or *nogroup*) groups are assigned to users without any

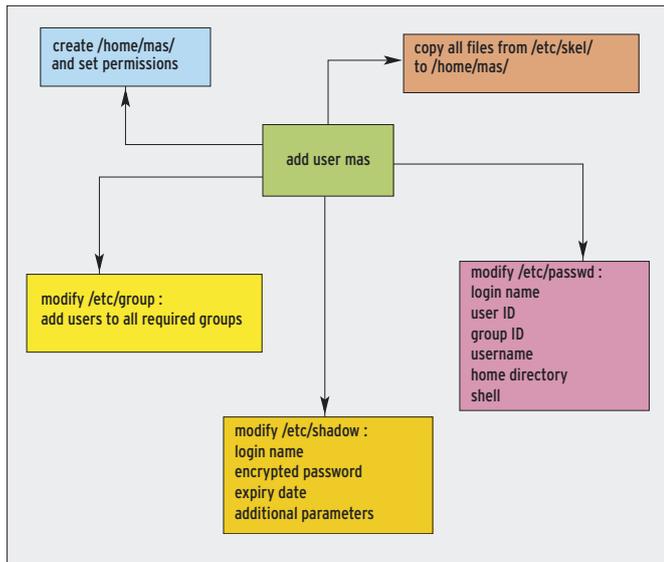


Figure 2: Creating a new user account involves a series of chores. Tools such as `useradd` offload most of the work off the administrator.

privileges, however, there is a danger of these groups mutating to “everybody” as software packages tend to create files with this group ownership. It makes more sense to set up a special nobody group for each software package.

Home Directories

When an administrator creates a new user, simply adding the home directory `/home/user` to `/etc/passwd` will not be a big help. Of course, the directory needs to be created. Besides the obvious `mkdir` step, the administrator has to remember to set appropriate user and group permissions by running `chown` (and possibly `chgrp`).

Depending on your distribution and preferences, you can set the permissions for home directories to 0755, 0711, or 0700. The first variant allows anyone read access to the directory. The second variant allows access to the files and directories below the directory, assuming that their permissions are set and their names are known, but preventing directory listings. The third variant prevents access by other users. As the home directory belongs to the new user, he or she can modify these defaults any time later.

But admins can’t just sit back and relax after creating the home directory. Defaults are needed for common programs and environmental variables need to be set up for the user. Most Unix programs use so-called dotfiles for configuration; dotfiles are hidden configuration

menus. Word processors and image manipulation programs need to support the local measurements and printer parameters. Browsers need to know where proxies are located, and so on.

The `/etc/skel/` (for skeleton) directory has a basic template for new home directories. Admins can store the dotfiles to be assigned to new user accounts in the skeleton directory. When a new home directory is created, the skeleton is simply copied to that directory.

This finally completes the user account and environment. Listing 1 shows you the commands needed to create a user account. Running this list of commands each time you need to create a new user is extremely time-consuming, so tools such as `adduser` and `useradd`

Listing 1: User Account for mas

```

01 $ su
02 Password:
03 # vipw
04 [...]
05 # passwd mas
06 # mkdir /home/mas
07 # cd /etc/skel && tar cf - . |
   (cd /home/mas && tar xpf -)
08 # chown -R mas:users /home/mas
09 # chmod -R u+rwX,go-rwx /home/
   mas
10 # exit
11 $
  
```

files (or directories) in the home directory belonging to the user running the program. It is easy to see where the name comes from, as the names of these files or directories always start with a dot.

The Skeleton

A window manager needs to know what software is installed to be able to provide appropriate

help automate these steps. Figure 2 gives an overview of the steps involved.

Advanced Settings

One big problem with pre-configuring new accounts is that many settings are unknown in advance. Although the local language and time zone are fairly easy to get right, and will tend not to vary, program specific settings are a completely different thing and can even change in the case of updates. To make things worse, administrators can’t assume that a user will immediately start to use the account and then take over managing the configuration options from that point onward.

In many cases it makes sense to set up permanent configuration files at a central location, assuming that the application in question supports this approach. Typical X11 window managers, or the `xinit` configuration are good examples of this. Administrators can simply maintain the central configuration structures in `/etc/X11/`. In case of an update, the administrator (or the package manager) simply modifies these directories to automatically propagate the new settings to users. If advanced users are unhappy with the defaults, administrators can assume that they will change the settings in their home directories.

Listing 2 shows an example of an `/etc/X11/xinit/xinitrc` file, which initializes the keyboard and profiles centrally (lines 11 through 17). Wherever users have made individual changes, the script automatically applies these changes (lines 19 through 25).

Admins can use a similar approach to set environmental variables in the central shell profiles `/etc/profile` (for Sh and Bash) or `/etc/csh.cshrc` and `/etc/csh.login` (for Csh and Tcsh). Wherever users have defined their own versions of these files in their home directories (again, these are dotfiles), the shell will run these files, assigning higher priority to the settings in these files, at least for variables. Central settings such as `ulimits` can be assigned to have priority over local variants. Bash allows

INFO

[1] John the Ripper, a password cracking tool: <http://www.openwall.com/john/>

administrators to define some variables as *readonly*. This gives administrators granular control over the settings.

If a centralized approach to configuring and managing accounts turns out to be impossible, because some applications do not support it and a workaround is not available, you can always resort to an old trick: replace the program file with a startup script.

The script will need to check the home directory for a version of the configuration file and, if this does not exist, copy the file with the default settings from a centralized location before launching the application itself. Depending on the amount of effort you are prepared to put into this, you could even tell the startup script to change the configuration files automatically in case of an update. ■

Listing 2: /etc/X11/xinit/xinitrc

```
01 #!/bin/sh
02 # $Id: xinitrc,v 1.2 2003/02/27 19:03:30 jharper
   Exp $
03
04 userresources=$HOME/.Xresources
05 usermodmap=$HOME/.Xmodmap
06 sysresources=/etc/X11/xinit/.Xresources
07 sysmodmap=/etc/X11/xinit/.Xmodmap
08
09 # merge in defaults and keymaps
10
11 if [ -f $sysresources ]; then
12     xrdp -merge $sysresources
13 fi
14
15 if [ -f $sysmodmap ]; then
16     xmodmap $sysmodmap
17 fi
18
19 if [ -f $userresources ]; then
20     xrdp -merge $userresources
21 fi
22
23 if [ -f $usermodmap ]; then
24     xmodmap $usermodmap
25 fi
26
27 # start some nice programs
28
29 xterm &
30
31 # start the window manager
32
33 exec fvwm2
```