User Accounts

Even a single-user workstation (Desktop Computer) uses multiple accounts. Such a computer may have just one user account, but several system accounts help keep the computer running.

Accounts enable multiple users to share a single computer without causing each other too much trouble.

User Accounts

The Users in a Linux system are stored in the `/etc/passwd` file. If you are on your own VM - somewhere near the bottom you should see yourself and joe.

On a brand new install you will see many users listed. Of course if you recall, we only added ourselves and joe. So what are all these other users for? These users are added because we don't want to give sudo power to all of our programs. So each program installed gets its own user with its own limited permissions. This is a protection for our computer.

User Info `/etc/passwd`

Examine the `/etc/passwd` file using `cat` or `less`. Here is what we are seeing:

- It is colon `:` separated. So each `:` denotes a new column.
- username
- password
  - The x is just a place-holder. The password is not really in this file
- User ID (UID)
  - Just like every computer needs a unique ip address, every user needs a unique id.

User Info `/etc/passwd`

- Group ID (GID)
  - Every user belongs to its own group, with its own group id
- Comment field
  - This is the Full name, phone number, office number, etc. It is okay if it is blank.
- Home Directory
  - This is the location of the users home directory - This is how we can know the full path of any users home directory - /home/smorgan vs /home/s/smorgan.
- Default Shell
  - When a user ssh's into our machine - this is where it is specified which shell they open. The Ubuntu default is /bin/bash. We can change our own shell and we can also disable a user's ability to ssh in.

Passwords

If passwords are not stored in the `/etc/passwd` file then where are they stored?

Passwords are actually stored in `/etc/shadow`. Try and view the contents of this file. Do you notice anything?

This file actually requires sudo power to view it. This is an obvious precaution. We don't want just anyone being able to see our passwords.

However, when you actually get to see inside the file - the passwords are encrypted. So not even the root user can see your password.

Passwords

Each line in the file is made up of the following:

- username
- password (encrypted)
- Date of the last password change
- Days until a change is allowed
- Days before change required
- Days of warning before pwd expires
- Days between expiration and deactivation (optional)
- Expiration date (optional)
- A reserved field (optional)

Passwords

If you are wondering how 17224 (or something similar) could possibly be a date. It actually refers to the number of days since January 1, 1970. So 17224 is actually January 27, 2017

- Try it: `sudo cat /etc/shadow | grep joe`
- Try it: `sudo chage -l joe`

LDAP

An LDAP (Lightweight Directory Access Protocol) Server is a Linux server that allows centralized user login / permission information. All the usernames and passwords are stored in a database. There are several reasons to have a centralized user database.

These are some of the advantages of an LDAP system

- Active login on many computers
  - Having to only remember one username and one password is a handy feature
- File/profile access on many computers
  - Accessing our files from any computer is great when someone else sits in our regular seat.
- Adding Users to a network
  - Imagine if a new student registers for class - What kind of effort would that take to manually add them to every computer. Imagine 10 computers, 20, 30, 100, etc.

LDAP (advantages)

- Changing your password on a network
  - Imagine if you forgot your password and had to change it. What would it take to change it on those 10, 20, 30, or 100 computers. What if you decided to only change it on the ones you actually use then one time someone else sits in your seat and now you are at a computer with the old forgotten password.
- Deactivating a user account
  - Imagine if every time a student graduates we had to deativate their account – one at a time on all 100 computers.

LDAP

The CIT department uses an LDAP Server to store the usernames, passwords and files of all the CIT students. DSU also uses a centralized user database. This is what allows us to log in to any computer with a single username and password. This is what allows us to see the same files on all computers once we've logged in.
Using an LDAP Server makes it easier to administrate mulitple users and computers on a network.

/etc/passwd with LDAP

Open a new terminal and log into scratch. Look at the /etc/passwd file. Look for your name. It's not there. The /etc/passwd file is for local users added on that machine only.

To view all users that have access to the system use

- `getent passwd`
- `getent passwd | grep joe`
- many times this gives us the same information as contained in `/etc/passwd`, but if using a centralized database for user accounts (as we do), you can get lots more information.
- Trying grepping getent passwd for your username on scratch.

/etc/passwd with LDAP

Try these additional commands on scratch

- `whoami`
  - Displays the username of the current user
  - This is helpful if your prompt doesn't tell you who you are and you can't remember if you logged in as joe.
  - There was a time when the prompt never displayed your username.
- `who`
  - Tells you who else is logged in to the computer
- `w`
  - Similar to who, but more verbose (more information)

Groups

- A user is always by default put into a group of the same name.
- Just like all users must have a unique id, all groups also have a unique id.
- Users can be in multiple groups.
- Why do we have groups?
  - Groups allow us to give unique permissions to a "group" of users.

The list of groups is stored in `/etc/group`

Groups

The file consists of the following:

- Group name
- password (not important here)
- Group ID (GID)
- Users in the group

The `groups` command will show us what groups we are members of

Groups

In Ubuntu there is a special group called the `sudo` group. This is the `sudoers list` and gives users of that group `sudo power` a.k.a `admin privileges`, `root access`, `superuser privileges` and anything else that might sound powerful.

- `cat /etc/group | grep sudo`

In distributions such as bsd and redhat the sudo group is called the `wheel` group. No matter the name, it has the same privileges.

High Level vs Low Level Commands

High level programs/commands provide an interface to assist you in creating new users and groups and they do a lot of things automatically for you. High level commands are most commonly used by administrators for one time changes.

Some High Level Commands

- adduser
- deluser
- addgroup
- delgroup

High Level vs Low Level Commands

Low level commands require that you set each option manually in the command line. If you fail to set the options then things like home directories and passwords may not be created. Low level commands are particularly useful for automated scripts that run without human interaction.

Some Low Level Commands

- useradd
- userdel
- groupadd
- groupdel

Adding Users and Groups

When you add a new user it will automatically create the user, create the required group of the same name, add all the required entries in `/etc/passwd`, `/etc/shadow`, and `/etc/group`, create the home directory, copy the default starter files from `/etc/skel`, prompt you for the new password and ask for the extra info about the user.

Just like everything else in Linux - usernames are usually lowercase.

Adding Users and Groups

When using the high level command everything is properly set up for the new user.

- `adduser frank`
- `cat /etc/passwd | grep frank`
- `cat /etc/group | grep frank`
- `ls -l /home`

What is frank's UID and GID? Who owns frank's home directory?

Adding Users and Groups

To add a group use the addgroup command

- `addgroup students`
- `addgroup friends`
- `cat /etc/passwd`
  - The students group is not there. It is not a user, it is only a group.
- `cat /etc/group | grep students`
  - Note the GID of students. It is the next available number.

Adding Users and Groups

Add another new user

- `adduser sally`
- `cat /etc/passwd | grep sally`
  - What do you notice about Sally's UID and GID?
  - Why don't they match? Because the students group used the next available GID without using a UID.
- `cat /etc/group`
  - See, Sally's GID is the next available one.

- `ls -l /home`

Adding Users To Groups

There are a few different ways to add users to a group. The easiest is using the addgroup command.

- The format of this command is
  - `addgroup user group`
- Try it:
  - `addgroup sally students`
  - `addgroup scott students`
  - `cat /etc/group`
  - `addgroup sally sudo`
  - `cat /etc/group | grep sudo`
  - `cat /etc/group | grep sally`
  - `groups sally`

Adding Users To Groups

The `id` command shows us not only our UID but also the GID of all our groups

- Try it:
  - `id`
  - `id sally`

Deleting Users and Groups

When you want to delete a user, by default Linux keeps the user's home directory and all the files they created.

- `deluser frank`
- `ls -l /home`

Frank's home directory still exists - Who owns frank's home directory now?

Nobody actually owns the frank directory now. It is owned by frank's UID and GID but Frank doesn't exist. This is why we have UID's and GID's because frank no longer exists but somebody has to own his old files.

Deleting Users and Groups

So what happens when we add a new user

- `adduser scott`
- `cat /etc/passwd | grep scott`
  - What do you notice about Scott's UID and GID?
  - Yes it is the same as Frank's old UID and GID.
- `cat /etc/group | grep scott`
- `ls -l /home`
  - Does Scott have his own home directory?
  - What happened to the frank directory? Who owns it now?

Deleting Users and Groups

If we wanted Scott to own Frank's files this is a good side-effect. But normally we do not want the newest user owning the long time employee's files.

If it causes this much trouble - why not have Linux autodelete the files when you remove a user?

- There might be something important in that user's files
- Other people might be actively dependent on those files

For that reason Linux doesn't automatically delete any files.

So the deluser command offers some options that allow us to choose what to do with those files as we delete the user.

Deleting Users and Groups

To see those options - check out the man page -

- `man deluser`

This is a great starter man page. It is simple and easy to read.

The first thing to remember while learning man pages is to not worry about reading or understanding everything. Let's hunt for just the things we want. The synopsis at the top show us how to use the command:

- deluser [options] [–force] [–remove-home] [–remove-all-files] [–backup] [–backup-to DIR] user

---

Deleting Users and Groups

This tells us that the command comes first and the user comes last, with options in the middle. It also tells us that we can use all of the above options in conjunction with each other. To know what these options do, scroll down the page and read the descriptions. Which options will help us remove the home directory of a user and which ones will help us make a backup of the files first?

From this page we learn that the best way to delete a user and remove his home directory that we don't care about is:

- `deluser --remove-home frank`

Deleting Users and Groups

Try it:

- `adduser jimmy`
- `cat /etc/passwd | grep jimmy`
- `ls -l /home`
- `deluser --remove-home jimmy`
- `cat /etc/passwd | grep jimmy`
- `ls -l /home`

Deleting Users and Groups

To delete a group is simple. But first let's add some users to the group we want to delete.

- `addgroup sally friends`
- `addgroup scott friends`

See that sally and scott are members of the friends group

- `cat /etc/group`
- `groups sally`
- `groups scott`

Deleting Users and Groups

Now delete the friends group

- `delgroup friends`
- `cat /etc/group`
- `groups sally`
- `groups scott`

You can see that the friends group is cleanly removed and all users that were part of the group are no longer part of the group.

Working as Other Users

- `sudo` - allows us to execute a single command as root
  - `sudo cat /etc/shadow`
- `su` - allows us to switch users, without any options or arguments we will switch to root.
  - `sudo su -` - When you have switched to the root user you will see a hash `#` symbol at the end of your prompt. This signifies that you are the root user.
  - `su - joe` - without sudo power it requires joe's password. `sudo su joe` - with sudo power it does not.
  - Type `exit` to return to your own user. It will return you to the directory you were in at the time you switched.
  - While it is possible to run the `su` command without the dash, it is rare to do so. To be safe when changing users, you should always run it like `su - joe` (with the dash)

Working as Other Users

- `login` - allows you to login as another user.
  - `sudo login joe` - This command requires sudo privileges. It also requires joe's password as well.
  - Notice that when you login as joe - It prints all the text just like when you ssh in as joe and you also switch to his home directory.
  - Type `logout` or `exit` to return to your own user. It will return you to the directory you were in at the time you switched.
- `ctrl+d` - ctrl+d will automatically execute the appropriate command: 'logout' or 'exit'.

Working as Root

Root user (different from `/` root directory) allows you to perform administrative tasks.

- Think of the motto - "Sudo Power is - All Power, All The Time, To Do All Damage!!!"
- There is nothing a root user cannot do.

Normal user accounts are restricted to doing only what is considered safe for the system. As a normal user you won't have the power to permanently break the computer. Most of your power is limited to breaking your own $HOME directory.

Working as Root

Before you ever use `sudo` ask yourself if you really need root access. Is this something that logically should require administrative rights?

- Sometimes there's a way to achieve your goal without superuser privileges such as using copy instead of move.
- Or most likely you aren't typing the command correctly and you need to check for typos.

Working as Root

- More cautions:

- Before pressing the Enter key after typing any command as root double check it! Linux doesn't ask "Are you sure?". It always follows your commands exactly as you type them.
- Never run a suspicious program as root. Most programs don't require root privileges to run. If you don't know exactly what it does - don't give it the power to act as root.
- Never actually `su` to root unless you have a whole string of commands that require sudo privileges.
  - Even then it is safer to type `sudo` just in case.
  - Use root privileges for as brief a period as possible then get back out to normal.
- Never leave a root shell available to others.
- Don't share root passwords.

More about passwords

- Selection of a good password is critical
  - What are some good password strategies?
- Passwords can be guessed by malicious individuals who know them or even who target them and look up personal information in telephone books, on Web pages, and so on.
- Although Linux encrypts its passwords internally, programs exist that feed entire dictionaries through Linux's password encryption algorithms for comparison to encrypted passwords. If a match is found, the password has been found.
- The best way to choose is password is to remember the key points that many systems require in your password.
  - 8 or more characters. The longer the password, the more difficult it is to guess.
  - Use Upper and Lower case. This adds 26 more guesses for each character.
  - Use numbers and symbols. 20 or more additional guesses for each character.
  - avoid personal, guessable passwords
  - avoid repeated or sequential characters
  - avoid common dictionary words
  - avoid runonsentences

More about passwords

Password privacy is absolutely important.

- The same password should not be used on multiple systems because doing so quickly turns a compromised account on one computer into a compromised account on all of them.
  - It can be tempting to use the same password for everything. However, this is risky. Sometimes it is helpful to have several passwords that you can use. So the loss of one doesn't compomise all of your logins.
- Writing passwords down or emailing them are both risky practices. Writing a password on a sticky note stuck to the computer's monitor is particularly risky.
- Spam e-mails sometimes try to trick users into revealing passwords by claiming that an email, banking, or other account has been deactivated or compromised.
- Users should never reveal their passwords to others, even people claiming to be system administrators—this is a common scam, but real system administrators don't need users' passwords.

More about passwords

Nobody ever, EVER needs your password. A root user has All Power, All The Time, To Do All Damage. A root user already has access to everything necessary. Keep your password private.

If someone claims to need your password, such as your web developer, because they don't have root access. First make sure you know and trust this person first - in most cases, a separate user can be created, a public/private key pair can be generated to allow temporary access without a password, or use can temporarily change your password to be a generic password that doesn't match any other login.

More about passwords

To change your password or the passsword of another user is simple.

- `passwd` - change your own password
- `sudo passwd joe` - change joe's password

File Permissions

- Files are owned by a particular user, and also a group.
- Permissions are set on a file that determine what can be done to is (or a directory).
- Can do with a GUI as well, but much more common via CLI.

File and Directory Permissions

There are three general classes of users:

- The user who owns the file ("User")
- Users belonging to the file's defined ownership group ("Group")
- Everyone else ("Other")

File and Directory Permissions

To see who is the owner, run the `ls -l` command

```
-rw-r--r-- 1 ralph admin 2558 Jan 8 07:41 filename
```

- -rw-r–r– represents permissions. drwxrwxrwx format. . rw- , r– , r–
- 1 – number of links
- ralph – Owner name (if user name is not a known user, the numeric user id displayed)
- admin – Group Owner name (if group name is not a known group, the numeric group id displayed)
- 2558 – number of bytes in the file (file size)
- Jan 8 07:41 – when file last modified
- filename – File name / pathname

Ownership

- Files and processes can be owned. I.e., the user that creates them is the default owner.
- The UID and GID are what are attached to a file.
- As root, we can change ownership of any file to any other user.
- As normal user, we can change the group ownership of a file that we own to be owned by another group that we are part of.

Ownership

- To change the ownership, we use the `chown` command. (change owner)
- Examples:
  - `chown vader wigs4me.txt`
  - `chown yoda I_am_green.png`
  - `chown yoda:goodguys rebelplans.doc` #change user and group
  - `chown :empire death_star_plans.txt` #just change group
- Or `chgrp`
  - `chgrp empire death_star_plans.txt`
- The `-R` option on both those commands is useful.

File Permissions

File ownership is meaningless without some way to specify what particular users can do with their own or other users' files. That is why we have permissions.

- View the output of `ls -l` again, and note the entries in the first column.
- Read allows viewing

- Write allows changing content
- Execute allows running a program (if it’s a file) or traversing a directory.

File Permissions

File Permissions (Symbolic Mode)

- We can use the symbols given in the output of `ls -l` when assigning permissions:
  - `chmod u=r,g=rw,o=rwx somefile.txt`
  - `chmod u+rw,o=rwx somefile.txt`
  - `chmod g+r,o+r somefile.txt`

File Permissions (Octal Mode)

- Octal notation is more compace than symbolic notation.
  - 4 = read permission
  - 2 = write permission
  - 1 = execute permission
- Then you can set permissions using this octal representation.
  - 7 = rwx, 6 = rw, 5 = rx
  - chmod 775 filename
- Setting permissions to none is done with a 0:
  - 640 = rw- r– — - user group others

Special Permission Rules

- Directories use the execute bit to grant permission to search the directory. This is a highly desirable characteristic for directories, so you’ll almost always find the execute bit set when the read bit is set.
- Permissions on symbolic links are always 777 (rwxrwxrwx, or lrwxrwxrwx, to include the file type code). This access applies only to the link file itself, not to the linked-to file. In other words, all users can read the contents of the link to discover the name of the file to which it points, but the permissions on the linked-to file determine its file access. Changing the permissions on a symbolic link affects the linked-to file.

More Special Permission Rules

Many of the permission rules don’t apply to root. The superuser can read or write any file on the computer—even files that grant access to nobody (that is, those that have 000 permissions). The superuser still needs an execute bit set to run a program file.

More examples

Without using octal mode, these are the rules:

- A code indicating the permission set you want to modify—u for the user (that is, the owner), g for the group, o for other users, and a for all permissions
- A symbol indicating whether you want to add (+), delete (-), or set the mode equal to (=) the stated value
- A code specifying what the permission should be, such as the common r, w, or x symbols, or various others for more advanced operations

---

What if?

- We assign ownership to a file/dir and then delete that user or group?
- We remove the execute permission on a directory?
- We create a symbolic link?
  - What is output of `ls -l`
  - Do permissions on link affect target?
  - What if I change permissions on target?
- How do we test execute permissions?
- We set permissions as 000. Can we access as root?

One more tidbit

A user mask or `umask` sets the default permissions on a file when it is created. The umask is the value that is removed from 666. So if the umask is 022, the files will all be 644. Directories will be removed from 777, so would be 755.

You can adjust the umask by using the `umask` command from the terminal. To make it persist, you would need to edit `.bashrc` or some such location.
