**4.2 Student Guide: Linux Access Controls**

**Class Overview**

In today's class, we will continue our introduction to Linux by covering one of the operating system's most important functions: access control. You will complete a sequence of tasks that will expose you to many of the most fundamental aspects of Linux security: password strength, careful control of the sudo command, and file permissions.

The skills and tools covered today will be essential for various professional roles, from systems administration to penetration testing.

**Class Objectives**

By the end of class today, you will be able to:

* Audit passwords using john.
* Elevate privileges with sudo and su.
* Create and manage users and groups.
* Inspect and set file permissions for sensitive files on the system.

**Lab Environment**

You will use your local Vagrant virtual machine for today's activities.

* Student access:
  + Username:sysadmin
  + Password: cybersecurity

**Slideshow**

The slides for today can be viewed on Google Drive here: [4.2 Slides](https://docs.google.com/presentation/d/1gywE-QHvHIWEGoqehxg8rO3GpiBqVflG5foGEf3qQ5g)

**01. Welcome and Review**

In the previous class, we covered:

* History and distributions of Linux.
* Navigating the Linux file structure using the command line.
* Managing processes with commands like top, ps, and kill.
* Installing packages using apt.

Today's class is focused on Linux's access control functions: passwords, file permissions, groups, and sudo access.

Access control refers to regulating what actions users and programs are and not allowed to take on a system.

Today will focus on users and the last day of the unit will focus on programs.

Today's class will build on Day 1 topics by covering:

* Passwords and password cracking.
* Switching between users with su.
* Elevating privileges with sudo.
* File permissions and access controls.
* Managing users and groups.

**02. Users and Passwords**

Your activities throughout the week involve auditing a malfunctioning server. Note the following:

* In the previous class, we viewed some important files, stopped a malicious process, and installed tools to facilitate the audit.
* Today, we will look at user access, starting with user passwords.
* In the previous class, we viewed the /etc/shadow file. This file contains hashes of every user's password on the system.
* The passwords are obfuscated with a hashing function, which means they are not stored in plain text.

**A Brief Introduction to Hashes and Password Cracking**

We have a whole unit later in the program dedicated to cryptography, where will cover hashing in depth. For now we will only cover a few basics. Note the following about hashes:

* A hash is a cryptographic function that takes data as input and translates it to a string of different, random-looking data.
* A hash will always output the same string for the same input data. So, when a password is entered into the system, the system hashes it the same way each time.
  + The same password will always produce the exact same hash.
* This hash is stored in the shadow file. When a user logs back in with the same password, the hash of the password they entered is compared with the hash stored for that user in the /etc/shadow file. If it matches, the user is logged in. If the hash doesn't match, the user is not logged in.

Note the following about password cracking tools:

* Password cracking tools do not reverse a password hash, but instead take a list of words and characters, and create a hash for each one.
* Each hash is then compared to the hash it is trying to crack. If the two hashes match, the password has been found.
* This form of password cracking is an example of a **brute force attack**.
* This is why the best passwords are long, with lots of random characters. The more random the password and the more characters it has, the longer it will take a cracking program to find a hash that matches it.

What makes a stronger password, it's complexity or it's length?

* Length and complexity work together to make a password strong.
* However, when it comes to brute force attacks, password length has more impact on the time it takes to crack it.

What is the current "industry standard" length for a password?

* Currently, the industry standard for password length is eight characters.
* We will soon find out that eight is only a minimum. It can still be cracked, though it takes some patience.

Navigate to [howsecureismypassword.net](https://howsecureismypassword.net/).

To see the importance of length, repeat the same character in one long string.

* Enter jjjjjjjj (eight j's). Eight characters crack instantly.
* Enter jjjjjjjjjjjj (twelve j's). Twelve characters will take four weeks to crack.
* Enter jjjjjjjjjjjjjjjj (sixteen j's). This password takes 35 thousand years to crack.
* Enter b4Ei@2! (seven random characters). This password only takes seven minutes to crack.
* Enter Jng0i$7w (eight random characters). This password takes nine hours to crack.
* Enter 534Yc8@CmF (ten random characters). This password takes 6 years to crack.
* Enter \*%uDiH2^T2n4 (twelve random characters). This password takes 34 thousand years to crack.

Takeaways:

* If a system requires sixteen characters and nothing else, the password will remain relatively strong, even if it includes words.
* Add a few extra characters and it gets exponentially more secure.
* In contrast, if you use all random characters, you *still* have to make the password at least 10 characters long for it to be very effective.

**Cracking Passwords**

* We just saw how and why passwords are hashed and stored.
* Modern password cracking software works using the following steps:
  + Takes a list of hashes as input.
  + Starts by hashing passwords from a given password list and comparing each hash to the list of hashes it was given.
  + If it matches a hash, it gives outputs of what password was used to create the hash.

This password cracking is a type of brute force attack because it will ultimately try *all* possible passwords, and eventually manage to reverse the hash.

* **John the Ripper** is a popular modern software because it can crack a wide variety of hashes.

**Using John the Ripper**

* The two steps needed to run John the Ripper are:
  + **Step 1:** Create a hashlist, which is a file that contains the hashes you are trying to crack.
  + **Step 2:** Run John the Ripper to crack the hashes.

Step 1: Creating the Hashlist

* **John The Ripper** can take an input file that contains usernames and password hashes.
  + John the Ripper requires input files to use a specific format.
  + Each line must look like: username:hash. A list of usernames and password hashes is often called a **hashlist**. In practice, it looks like:
* admin:e08e4506d2e3f370a5e8ab79647df309

guest:a132mj06d2e3f370a5e8ab79647df309

* You can also simply grab one whole record from the /etc/shadow file to add to the hashlist.
  + For example:

sally:$6$c0QGG1OFuiDGNKZT$wzbxLSWFOSyeSiyNZc2wNjaKr1B/w.D1xp7QBU0wG6xbBUbdZKEb1HwmW2Zn92/9jbVd.slXMByeLJeh1btOD.:18387:0:99999:7:::

* + John the Ripper knows how to grab the hash from a shadow file record.

Step 2: Running John the Ripper

* You run John the Ripper with the following simple format:
  + john <hashlist>.
    - For example: john hashlist.txt
* You can also have John the Ripper run against a predefined wordlist to speed up the cracking process with the following format:
  + john <hashlist> –wordlist="wordlist.txt"
    - Note that while a smaller wordlist may speed up the cracking process, if the wordlist doesn't contain the password, it will not be cracked.
  + There are many large wordlists available on the internet, but our distribution comes with a popular wordlist called rockyou.txt
    - This wordlist is located in the /usr/share/wordlists/ directory
    - An example command to run with this wordlist looks like the following:
      * john hashlist.txt –wordlist=/usr/share/wordlists/rockyou.txt
  + This process can take a very long time on real files (hours, days, or even weeks or months), but the passwords in today's exercise should break quickly.
* You can see the passwords that john has already cracked by running:
  + john --show <hashlist>.

**03: Activity: Talk to John**

* [Activity File: Let's Talk to John](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/2/Activities/03_Talk_to_John/Unsolved/README.md)

**04: Activity Review: Let's Talk to John**

* [Solution Guide: Let's Talk to John](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/2/Activities/03_Talk_to_John/Solved/README.md)

**05: Privileges, root, sudo and su Demo**

We've used sudo for several commands in the last two days of class.

* Every file and program on a Linux system has permissions associated with it. These permissions tell the system which user can access that file or run that program.
* Additionally, administrators can place users in a group, and set file and program permissions to allow a specific group or groups to have access.
* For instance, a company can create a group for employees who work in Marketing and another group for employees who work in Accounting. The administrator can give these groups access to specific programs needed by their department.
* The permissions for a given file or program apply to all the users on the system, except for the root user.
* The root user is the super user, or the highest administrator on the system. The root user has complete access to the system and can perform any action, access any file, and run any program.

During the last activity, we learned just how sensitive the /etc/shadow file is. Typically, **only** the root user has access to this file. This is an example of how permissions can protect parts of the system.

When an attacker is trying to gain access to a system, they are usually trying to gain root access, or access to the root user, so they can do whatever they want to the system. Hackers can achieve this access by switching users.

**Switching Users and Elevating Privileges**

The Linux system can access different users with su:

* su stands for "switch user." If you have another user's password, you can log in as that user with su <username>.
* Switching users can be helpful for troubleshooting. You can see firsthand what the user has access to, and test their permissions. You can also see what they've been doing with the system from their perspective, with full access to their files.
* From a security perspective, switching users allows you to use the system with their permissions. This lets you run commands as that user, view files that only that user has access to, and otherwise imitate the user.

Linux systems secure root access with sudo.

* Properly secured Linux systems do not allow anyone to log in as the root user on the system. Instead, following the principle of least privilege, if a user needs access to something only the root user can do, they can use the sudo command to invoke the root user just for that one command.
* sudo stands for "superuser do," and if a normal user is allowed to use sudo, they can run a root-privileged command. When the command is complete, the user is reverted to their normal access.
* sudo can also control which commands the user can run as root user. This way, the system has granular control over who can run root commands, and which ones. It also keeps a log of exactly which user runs which commands, which can be reviewed as needed.

Note the following about configuring sudo access:

* sudo access is configured using a configuration file, the sudoers file.
* Inside that file, a sudo group is specified along with which commands sudo can be used with.
* The sudo group is typically given full system access to use sudo.
* Any user on the system that needs sudo access is then added as a member of the group. Any member of the group receives the same access and ability to use sudo for any command.
* Alternatively, a user can be kept out of the sudo group and added to the configuration file individually, along with a specification of which commands that user can use sudo for.
* Adding a user individually to the sudo configuration file is common when the sysadmin has a user that only needs sudo access for a few commands.

**su vs. sudo Demonstration**

In the next demo, we will attempt to update all of our existing software packages.

* If our privileges do not allow us to do so, we will first use su to switch directly to the root user.
* We'll then show the dangers of working directly as the root user.
* We'll then do the same updates by using sudo instead and show why this is the more secure option.

We will use the following commands to do these tasks:

* whoami to view your current user.
* su to switch to another user, in this case, the root user.
* sudo to invoke the root user for one command only.
* Run whoami to show that you are the instructor user.
* Run apt update and note that this doesn't work.
* # apt update
* Reading package lists... Done
* E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)

E: Unable to lock directory /var/lib/apt/lists/

* Only the root user has the ability to use the apt program. Because we are not root, we received a Permissions denied error.
* The Permission denied message indicates that you do not have permission to open a file (/var/lib/apt/lists/lock), and that you need to run apt with elevated privileges.

We will now log in as the root user with su, which again, stands for "switch user."

* Run sudo su (password: instructor)
* Run whoami to show that we are now root.
* The prompt now also uses a #, indicating that you are the root user. A standard user's prompt will generally show a $.

Now that we’re the root user, we can install packages.

* Run apt update as root and we should see the following:
* Hit:1 http://us.archive.ubuntu.com/ubuntu bionic InRelease
* Get:2 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
* Get:3 https://download.docker.com/linux/ubuntu bionic InRelease [64.4 kB]

Get:4 http://us.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]

* This command updates the package repositories so we can download the latest software.

Once you, or a process, is logged in as root, you can make any changes you want to the system, including changes or malicious actions that may harm the system.

We will perform a quick example:

* Make sure you're (still) root with sudo su (password instructor) and do the following:
  + Run ls /home to show the current home folders.
  + Run rm -r /home/john to remove the home folder for john.
  + Run ls home again to show that it is removed.
  + Run mkdir /home/john to create a new empty home directory for john.
  + Lastly, run chown -R john: /home/john to give the user, john, ownership of their newly created home directory.

At no time during this process were you asked for a password.

* That is because you are the root user and you can perform any action you want, without the system stopping you.
* Not only is this a problem if you want to make system and software changes, but it's also a problem if you were to make a mistake and remove the wrong files.

Once you are logged in as another user, you can log out by typing exit.

* Run exit. This will log you out from the root user.

A better way to make the same changes is to use sudo.

* sudo usually and preferably prompts the user for a password, and it will only allow you to complete the actions you have access to.
* sudo adds a layer of security because it forces the administrator to consciously run a command with privileges.
* sudo also saves a log for each time the command is used. Therefore, an administrator can audit the log to find out which user did what.

Run sudo apt update. We have to first enter our password to use sudo.

Now we will try to delete a directory like we did previously:

* Run ls /home to show the current home folders.
* Run rm -r /home/john

Your output should be similar to:

$ rm -r /home/john

rm: cannot rm directory ‘/home/john’: Permission denied

Restricting sudo use among users and only allowing access for specific commands reduces the risk of harm to the system.

**Assigning sudo Access Demo**

In the previous demo, we didn't have to log in as root because we could just use the sudo command to use the apt command.

If we want to see exactly what sudo access we have, we can run sudo -l.

* Run sudo -l. Your output should contain the line:

User instructor may run the following commands on localhost:

(ALL) ALL: ALL

* We, the instructor user, have ALL access.

We can check the privileges of a user with the -lU options. We can check the privileges of a user, sally, with the following command.

* Run sudo -lU sally. The output should be:

User sally is not allowed to run sudo on ubuntu-vm.

We can give sally full sudo access by adding her to the sudo group.

* Run sudo usermod -aG sudo sally to add Sally to the sudo group.
* Run sudo -lU sally. Your output should now read:

User user may run the following commands on localhost:

(ALL) ALL: ALL

* sally now has full access.

We can also give a user sudo access for just a single update. For example, we want to give our user john sudo access for apt so he can run software updates.

Why can't we just add john to the sudo group?

* This will give him full access to run any command, which we don't want.

Remember, the sudo settings are configured in the /etc/sudoers file.

To update the /etc/sudoers file, you must use the command visudo, which opens the etc/sudoers file using Nano.

* Using visudo to edit this file is necessary because visudo does a syntax check on the sudoers file before it is saved, to prevent corruption of the file.
* Breaking this file can lock you out of the system entirely, so you want to be sure to always use visudo to edit the file.
* ⚠ **Troubleshooting Help**: If you break this file and get locked out of using sudo, review the following thread:
  + [Ask Ubuntu: How to Modify an Invalid etc Sudoers File](https://askubuntu.com/questions/73864/how-to-modify-an-invalid-etc-sudoers-file).

Run sudo visudo.

Scroll down to find the following lines at the bottom of the file:

# User privilege specification

root ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges

%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command

%sudo ALL=(ALL:ALL) ALL

* root ALL=(ALL:ALL) ALL: Allow the root user to run any command under any user or group on any system.
* %admin ALL=(ALL) ALL: Allow all members of the admin group to run any command with sudo under any user on any system.
* %sudo ALL=(ALL) ALL: Allow all members of the sudo group to run any command with sudo under any user on any system.

The general syntax of these lines are as follows:

* [USER or %GROUP] HOST=(USER:GROUP) COMMAND
  + The HOST is normally set to ALL but can be changed if the administrator wants to limit which machines can use this file.
* For a group entry, the GROUP inside the parenthesis can be left out: %GROUP HOST=(USER) COMMAND
* To remove the password requirement, NOPASSWD is added: USER HOST=(USER) NOPASSWD: COMMAND

This means any user in the secondary group sudo *or* admin can use sudo to run privileged commands with their password.

Add the line john ALL=(ALL:ALL) /usr/bin/apt.

* This allows the user john to run the apt command with sudo as the root user, on any host after entering his password.
* john now has access to run the apt command and update software packages.

Save and exit.

Verify your new settings.

* Run sudo -lU john
* Output should look like:
* User john may run the following commands on localhost:

ALL=(ALL:ALL) /usr/bin/apt

**Attackers Gaining Root Access Demo**

Even when sudo use is restricted to specific commands, depending on the command, a user can still gain access to root.

In this last demo, we will pretend we are an attacker and attempt to gain root access from an account that has sudo access to one of these commands.

There are several different commands for which this can be a problem, but today we will look at the less command.

Let's return to our example of john. In addition to allowing john the ability to install software, we may also want to allow him to read any of the sensitive files on the system using less.

* less, however, has a feature that allows you to run commands without exiting the less command, and those commands are run with the same privileges that less has.
  + In other words, if a user has sudo access for less, they can open less and then start running commands from inside less with sudo privileges.

To demonstrate this, we will use sudo less with the admin user and then move from less directly into a root shell with !bash.

* Run sudo visudo and edit the entry for john to give him sudo access to less.
* Output should look like:

john ALL=(ALL:ALL) /usr/bin/apt, /usr/bin/less

Save and exit.

Now we will run sudo less on any file that belongs to John:

* Run su john (password: lakers) to switch to John's user.
* Run touch /home/john/my\_file
* Run sudo less /home/john/my\_file

Because we ran less with the sudo command, when we are inside less we are no longer john. Instead, we are now root.

To run a command from inside less, we use ! followed by the command.

We can run any command with less, but it makes the most sense to run bash. This command will launch another shell from inside less with your current root privileges.

* Type !bash and press Enter to drop into a root shell.

We now have a # at the prompt again, indicating we have root privileges.

* Run whoami to confirm that you are root.

Attackers often look for this kind of loophole to escalate their privileges on a system.

* This kind of exploit is called an **escape** exploit because you are escaping the program less and getting full system access.
* It's important to restrict which commands users can use with sudo in /etc/sudoers and to always make sure there are no known vulnerabilities with the commands you *do* allow.

**Summary**

* whoami to determine your current user.
* su to switch to another user, in this case the root user.
* sudo to invoke the root user for one command only.
* sudo -l to list the sudo privileges for a user.
* visudo to edit the sudoers file.

**06: Activity: sudo Wrestling**

* [Activity File: sudo Wrestling](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/2/Activities/06_Sudo_Wrestling/Unsolved/README.md)

**07. Activity Review: sudo Wrestling**

* [Solution Guide: Sudo Wrestling](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/2/Activities/06_Sudo_Wrestling/Solved/README.md)

**08. Break**

**09. Users and Groups**

We will now discuss users and groups in more depth. As a quick review:

* Linux is a multi-user OS and related users can be added to groups.
* We briefly discussed this when we spoke about the sudo and admin groups.
* In the case of sudo, all users added to the sudo or admin groups have full access to sudo.

Linux has the ability to create groups of users for other functions like file or services sharing.

* If a company has different departments like Sales, Accounting and Marketing, a Linux administrator can create a group for each department. Only the users in the group can access files owned by the group.
* Therefore, a system admin must know how to to add and remove users from a system, add and remove groups, and add and remove users from those groups.

Linux has a few easy commands that are used specifically for user and group management, which we will focus on in this section.

Before diving into these commands, we’ll cover how Linux identifies users and groups in the system using the id command.

* Linux associates a specific number to each user for identification purposes. This number called a **user ID** or **UID**.
* When Linux needs to identify a user, it doesn’t look at the username, it uses the UID number.
* System users, or automated users designed to complete system tasks, have UIDs assigned at numbers less than 1000.
* Standard users, or users that are assigned to a real person, have UIDs assigned at numbers greater than 1000.
* The root user always has the UID of 0.
* Likewise, the **GID** is the **group ID** that is associated with a group.
  + Our UID is above 1000, which indicates that we are a standard user.

We can also see the UID for each user in the /etc/passwd file.

* Run head /etc/passwd
  + The system UIDs start with root at 0, and move up from there.

UIDs and GIDs are only a system number that Linux uses for identification. If we want to see the groups that a user belongs to, we can use the command groups.

* Run groups
  + Note that it prints your user's groups to the screen.
* Run id.
  + This also shows us the groups along with the GIDs assigned to them.

**Users and Groups Demo**

In the upcoming demo, we’ll dive into more actions around user and group management, using the following scenario:

* The company you work for recently had a change to its developer team. Mike, a lead developer, has left the company. Joseph has joined as a new junior developer.
* The company's Linux system has never been set up properly with a developers group. Instead, Mike was part of the general group.
* As the sysadmin for this system, you need to remove Mike from the general group, remove the general group, and delete Mike's user from the system. Then, you need to add Joseph to the system, create a developers group, and add Joseph to this group.

To accomplish these tasks, you will:

1. Get group info for Mike's user using the command groups.
2. Lock Mike's account to prevent him from logging in using the command usermod.
3. Remove the mike user from the general group with the command usermod.
4. Delete the mike user by using the command deluser --remove-home.
5. Delete the general group using the command delgroup.
6. Create a joseph user with the command adduser.
7. Create a developer group using the command addgroup.
8. Add the joseph user to the developer group using the command usermod.

First, we'll see what groups Mike belongs to.

* Run groups mike to print Mike's groups to the screen.
* Your output should be:
* $ groups mike

mike general

Each user is also a member of a group that shares the name of the user.

* When a user is created, Linux by default creates a group of which that user becomes a member.

The usermod command has many different options and lets us do many things to user accounts, but we are going to look at the -L and -G options. The -L option will lock the account and the -G option will specify the groups a user should belong to.

* Run sudo usermod -L mike to lock the account.
  + sudo: Only root can modify users and groups, so we will have to use sudo for all of our commands.
  + usermod: Allows us to make many modifications to users. In this case, we are using it to add and remove groups.
  + -L: usermod flag that locks an account so it cannot be logged into.
  + mike: The usermod command always ends with the user we are modifying.
* Run sudo usermod -G mike mike to remove mike from the general group.
  + sudo: Only root can modify users and groups, so we will have to use sudo for all of our commands.
  + usermod: Allows us to make many modifications to users. In this case, we are using it to add and remove groups.
  + -G: This usermod flag specifies which groups the user should belong to. The groups that we specify following this command will be the *only* groups that user belongs to after we run the command.
  + mike: Following the -G flag are the groups we want the user to belong to. In this case, we want the user mike to be a member of the mike group only, effectively removing the general group.
  + mike: The usermod command always ends with the user we are modifying.
* Run groups mike to confirm the result.
  + Your output should be:
  + $ groups mike

mike

* + Mike has successfully been removed from the general group.

We can now remove the Mike user from the system using the deluser command.

* Run sudo deluser --remove-home mike
  + sudo: Only root can modify users and groups, so we will have to use sudo for all of our commands.
  + deluser: Allows us to delete users from the system.
  + --remove-home: deluser flag that removes the user's home folder along with the user.
  + mike: The deluser command always ends with the user we want to delete.

If we use the deluser command without any flags, it will leave all of Mike's files intact including his home folder. In this case, we will remove the user *and* all of his home folder files.

* Run ls /home to confirm your results.
  + Mike's home folder has been deleted.

Do you remember how to verify users or groups on the system?

* You can check for users in the /etc/passwd file with grep <user name> /etc/passwd.
* You can check for groups in the /etc/group file with grep <group name> /etc/group.
  + Run grep mike /etc/passwd to verify that mike is deleted.
  + Run grep general /etc/group.
  + The general group still exists. The line for the general group should look like this:

general:x:32:

* + If this group had any members, they would be listed after the last colon (:). In this case, there are no members left in this group.

We can now remove the general group with the delgroup.

* Run sudo delgroup general
  + Run grep general /etc/group to verify it is gone.

Now we will create our new user, joseph.

* Run sudo adduser joseph and complete the prompts to give joseph a password and other info.
* Run groups joseph to display the joseph group.

Your output should look like:

$ groups joseph

joseph

Remember, when a user is added to the system, by default a group by the same name is added. Also, when a user is deleted, their group is also deleted, as long as no other users are members of that group.

Next, we will create a new developer group using the addgroup command. Then we can add the user joseph to the group.

* Run sudo addgroup developers

We received a Done message, but we can also verify this group was added in the /etc/group file.

* Run tail /etc/group
* **Note:** Since our new groups will be the last line in the group file, using tail is easier and quicker than grep in this case.
  + We can now see both the new joseph group that was created when we added the user joseph, as well as the new developers group.

We are now ready to add joseph to the developers group using the usermod command.

* Run sudo usermod -aG developers joseph
  + sudo: Only root can modify users and groups, so we will have to use sudo for all of our commands.
  + usermod: Allows us to make many modifications to users. In this case, we are using it to add and remove groups.
  + -aG: This usermod flag combination (*add group*) specifies which groups the user should be added to.
  + developers: Following the -aG flag are the groups we want to add the user to. In this case, we want the user joseph to the developers group.
  + joseph: The usermod command always ends with the user we are modifying.
* Run groups joseph
  + Joseph is now part of the developer group as a secondary group.
* Your output should be:
* $ groups joseph

joseph : joseph developers

* We have now created the joseph user, created the developers group and added joseph to it.

A user always has a primary group that is typically the same name as the user. The primary group can be changed to another group, but there isn't usually a reason to do so.

Groups that a user is added to beyond the primary group are known as secondary groups. A user can be a member of unlimited secondary groups.

Summary:

1. Received group info for the mike user with the command groups.
2. Used usermod to lock the mike user account to prevent it from logging into our system.
3. Removed the mike user from the general group using the command usermod.
4. Deleted the mike user by using the command deluser --remove-home.
5. Deleted the general group using the command delgroup.
6. Created a joseph user by using the command adduser.
7. Created a developer group using the command addgroup.
8. Added the joseph user to the developer group using the command usermod.

**10. Activity: Users and Groups Activity**

* [Activity File: Users and Groups](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/04-Linux-SysAdmin-Fundamentals/2/Activities/10_Users_and_Groups/Unsolved/README.md)

**11. Activity Review: Users and Groups**

* [Solution Guide: Users and Groups](/University-of-Minnesota-Boot-Camp/uofm-virt-cyber-pt-06-2021-u-lol/-/blob/master/Activities/10_STU_Users_and_Groups/Solved/README.md)

© 2020 Trilogy Education Services, a 2U, Inc. brand. All Rights Reserved.