Linux system administration encompasses a wide range of tasks focused on maintaining and managing Linux-based systems. Two major tasks here are user and group management and automation of repetitive tasks.

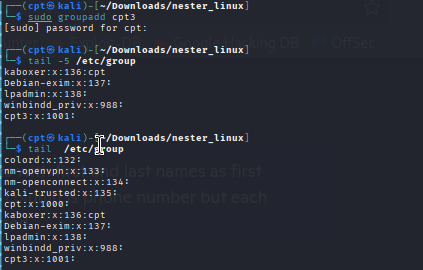
User and Group Management:

Adding, deleting, and managing user accounts, assigning permissions, and managing group memberships are fundamental tasks.

Automation:

Automating repetitive tasks using shell scripts, cron jobs, and other tools helps streamline administration and reduce errors.

***groupadd - to create a new group in Linux***



As we can see new group id (1001) was assign to the new group.

If admin want to have a different id, key “g” could be used.

*sudo groupadd -g 1001 developers*

***groupdel***

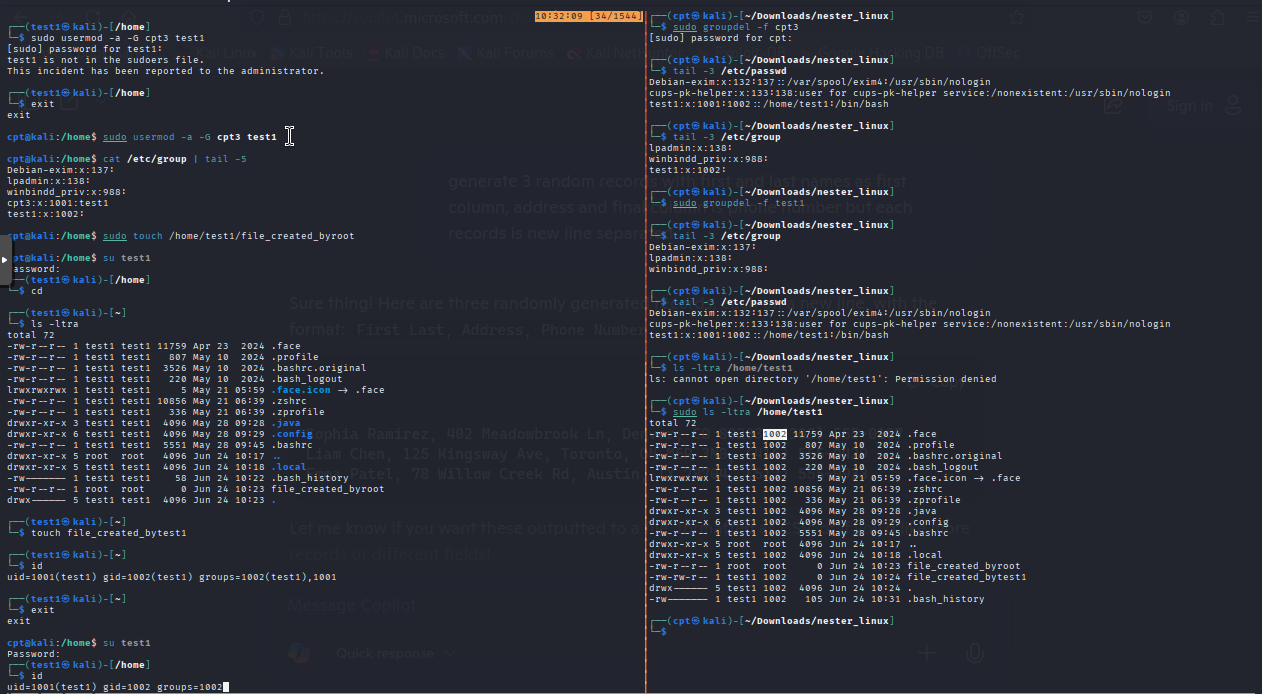
Interesting fact is that command below SHOULD forcefully delete a group, including removing all users associated with it.

*sudo groupdel -f example\_group*

Even when a group is a user's **primary group**, running sudo groupdel -f example\_group doesn't necessarily remove the user. Here's why:

* **The force (**-f**) flag** only suppresses warnings—it doesn't magically resolve underlying conflicts. If a user’s primary group is the one you're deleting, groupdel is supposed to fail unless you reassign the user's primary group before deletion.
* **But** on some systems, if the group’s entry is forcibly removed anyway (especially with manual edits to /etc/group), the user account might still technically exist—but now their GID (group ID) points to a non-existent group. That can result in an inconsistent or broken state for that user. It’s not best practice, but it can happen.

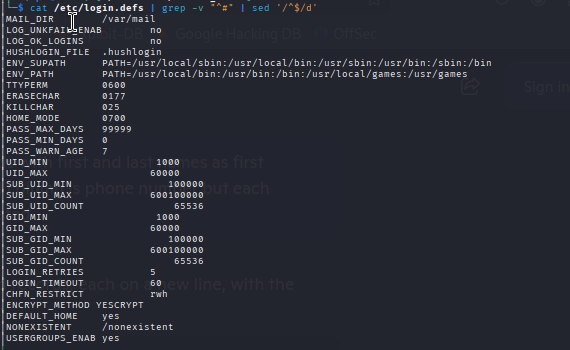
**See below new user with primary group is added to supplementary group, but after deleting BOTH groups, user still exists but all files in home have missing group for ownership except former group id 1002.**

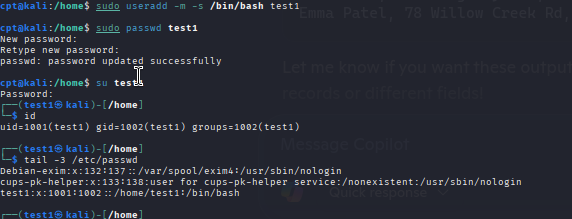


**useradd -** To add a new user.

When we add a new user in Linux with the ‘useradd‘ command, it gets created in a locked state. To unlock that user account, we need to set a password for that account using the ‘passwd‘ command.

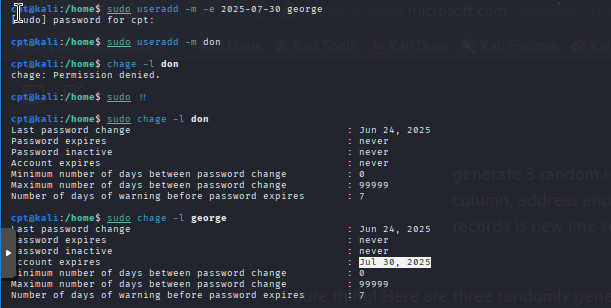
By default, the useradd command in Linux does not automatically create a home directory unless you specifically tell it to (with help of “m” key)—or unless your system is configured to do so. Configuration in ***/etc/login.defs***: There's a setting called ***CREATE\_HOME***. If it's set to no, even using useradd without -m won’t make the home directory. You can check and update it if you want that behavior by default.



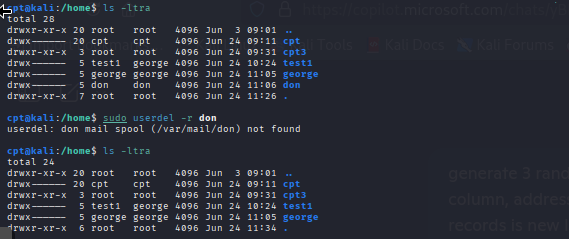


By default, when we add users with the ‘useradd‘ command, the user account never expires, meaning their expiry date is set to 0 (which means never expired). However, we can set the expiry date using the '-e' option, which should be in the YYYY-MM-DD format. This is helpful for creating temporary accounts for a specific period of time.

*useradd -e 2025-08-27 george*



To delete the user INCLUDING the home folder, key “r” can be used.



/etc/skel is a subtle but powerful piece of the user creation process in Linux.

It’s a **template directory**. Think of it as a skeleton (hence the name “skel”) for user home directories. When you create a new user with the -m option (like sudo useradd -m username), the contents of /etc/skel are automatically copied into the new user’s home directory (/home/username).

By default, it typically contains things like:

* .bashrc
* .profile
* .bash\_logout

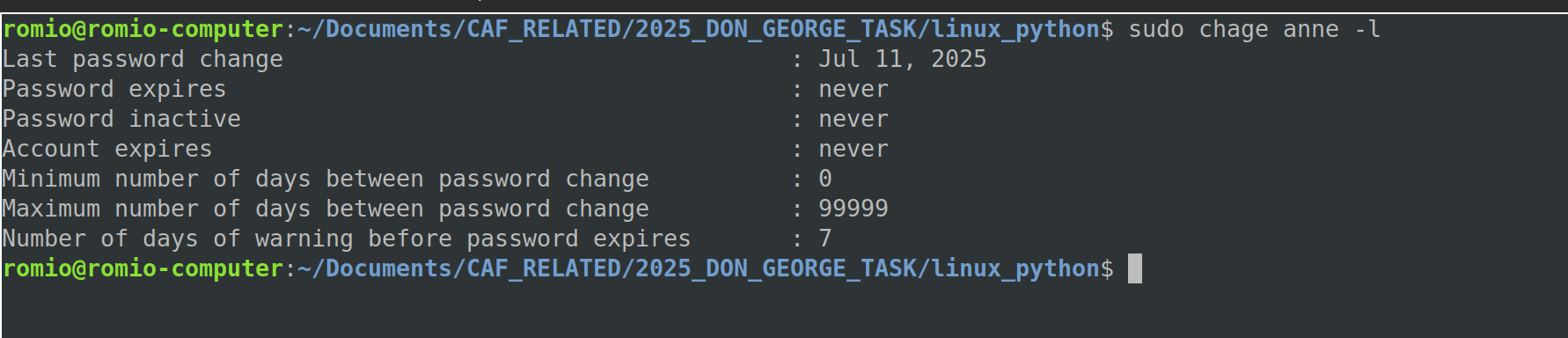
These are standard configuration files that customize the shell environment when the user logs in.

But here’s where it gets interesting: you can **add your own files** or scripts to /etc/skel, and they’ll be included in every new user’s home folder. Want everyone to get a README.txt or a fun welcome script? Pop it into /etc/skel, and voilà—every new user will have it from day one.

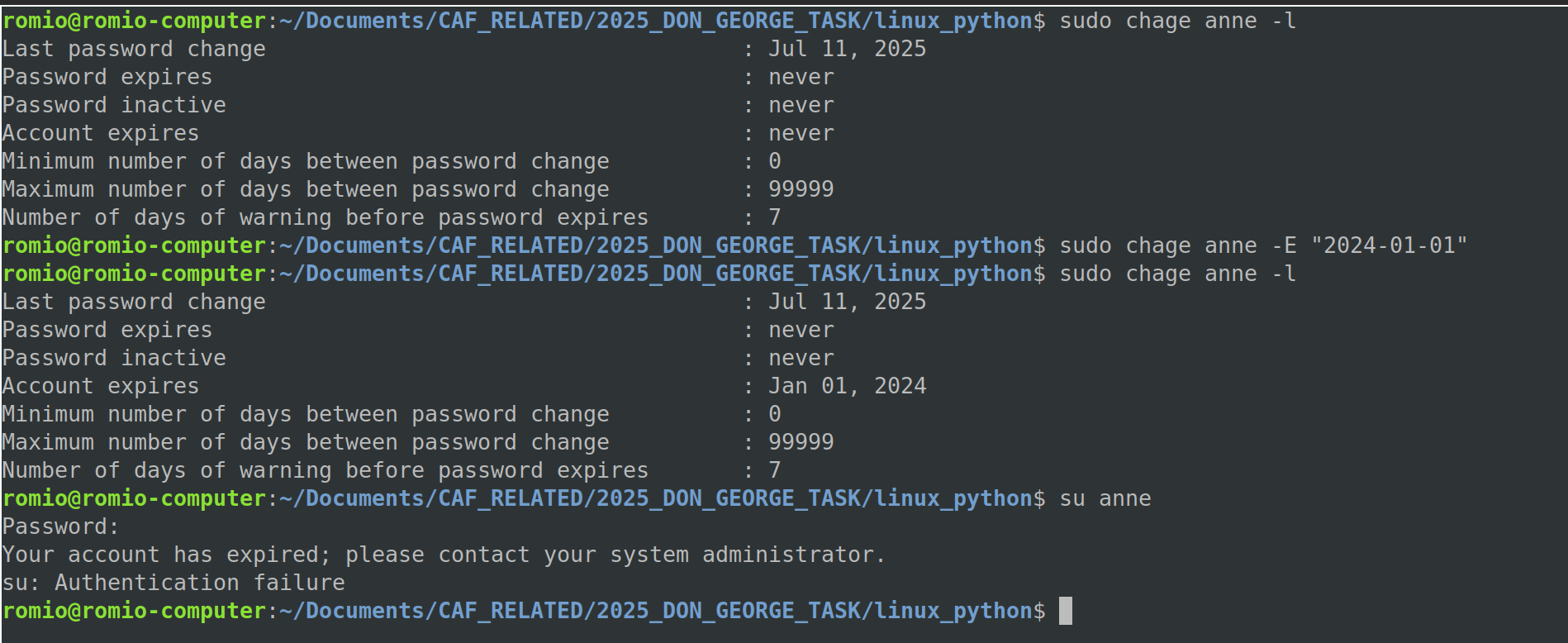
**chage**

The '****chage'**** command in Linux is a powerful tool used to manage user password expiry and account aging information. It is particularly useful in environments where user access needs to be controlled over time, such as when login access is time-bound or when it’s necessary to enforce regular password changes for security purposes. With the '****chage'**** command, administrators can view and modify password expiry details, set mandatory password change intervals, specify account expiration dates, and more.

Key **-l** to list current settings



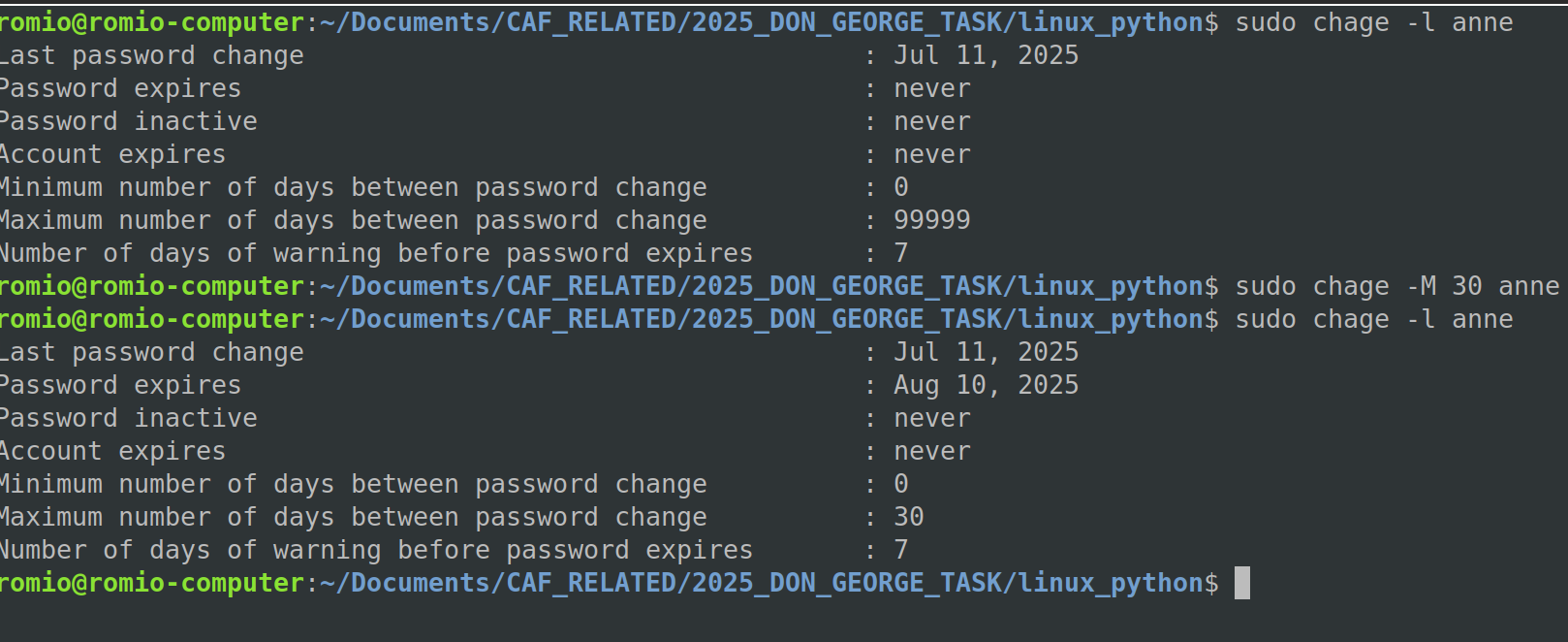
### ****'-E'** option** (Set Account Expiry Date). In the example below, we assign expiring date in the past. In result account is imitatively got deactivated.



The following command removes an expiration from an account:

*sudo chage -E -1 anne*

### ****'-M'** and **'-m'** option** (Set Maximum and Minimum Password Age)



### ****'-d'** option** (Set Last Password Change Date)

BUT if we use the command below, where instead of date we use 0, this way we can make the user to change his/her current password on next authorization attempt:

*sudo chage -d 0 anne*

### As you can notice, retyping old password is not allowed.

### ****'-I'** option** (Set Inactivity Period After Password Expiry)

Use this option to specify the number of days the account should be inactive after its expiry. It is necessary that the user should change the password after it expires, this command is useful when the user does not login after its expiry. Even after this inactivity period if the password is not changed then the account is locked and the user should approach the admin to unlock it.