Introduction to Operating Systems

5. Users and Permissions

2026-11-03

USO Team

Bibliography

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 2 - *Basic Commands and Directory Hierarchy*
     - Section 2.17 - *File Modes and Permissions*
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.3 - *User Management Files*
     - Section 7.4 - *getty and login*
     - Section 7.9 - *User Access Topics*
     - Section 7.10 - *Pluggable Authentication Modules*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

We trust you have received the usual lecture from the local System Administrator. It usually boils down to these three things:

#1) Respect the privacy of others.
#2) Think before you type.
#3) With great power comes great responsibility.

`sudo`, the first time you use it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Users

- Users
- The Users Database
- Local Password Storage
- Groups
- Logging In
- The `root` User
- Becoming `root`
- File Permissions
- Special Permissions
- The Label of a Process
- The Prompt
- Managing Users and Groups

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Users

the label every process runs under

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.3.1 - *The /etc/passwd File*
     - Section 7.9.2 - *Process Ownership, Effective UID, Real UID, and Saved UID*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

What Is a User?

👤 for the kernel, a user is just a **number**: the **UID**

- a user has a **name** for humans, `alice`, and a **number** for the kernel, the **UID** (*User ID*), `1000`

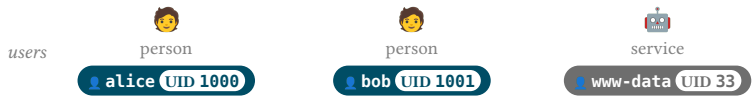


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

What Is a User?

👤 for the kernel, a user is just a **number**: the **UID**

- a user has a **name** for humans, `alice`, and a **number** for the kernel, the **UID** (*User ID*), `1000`
- not only people: each service has its own user, `www-data` is UID `33`

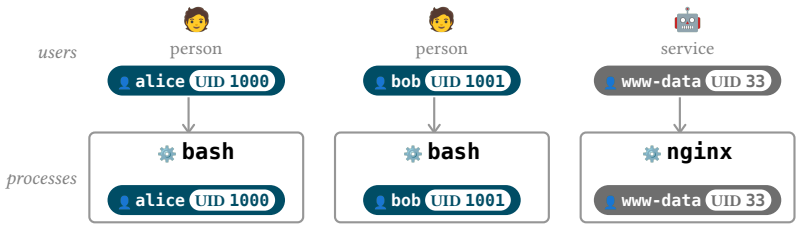


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

What Is a User?

👤 for the kernel, a user is just a **number**: the **UID**

- a user has a **name** for humans, `alice`, and a **number** for the kernel, the **UID** (*User ID*), `1000`
- not only people: each service has its own user, `www-data` is UID `33`
- every **process** carries the **UID** of its user, as a **label**, all its life

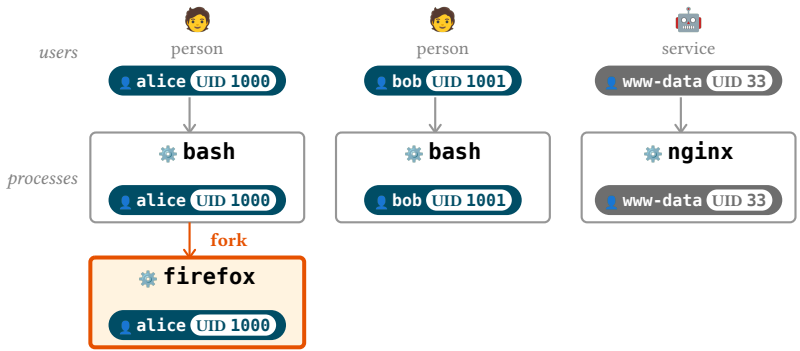


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

What Is a User?

👤 for the kernel, a user is just a **number**: the **UID**

- a user has a **name** for humans, `alice`, and a **number** for the kernel, the **UID** (*User ID*), `1000`
- not only people: each service has its own user, `www-data` is UID `33`
- every **process** carries the **UID** of its user, as a **label**, all its life
- a child **inherits** the UID: `firefox`, started by `alice`'s `bash`, also has UID `1000`

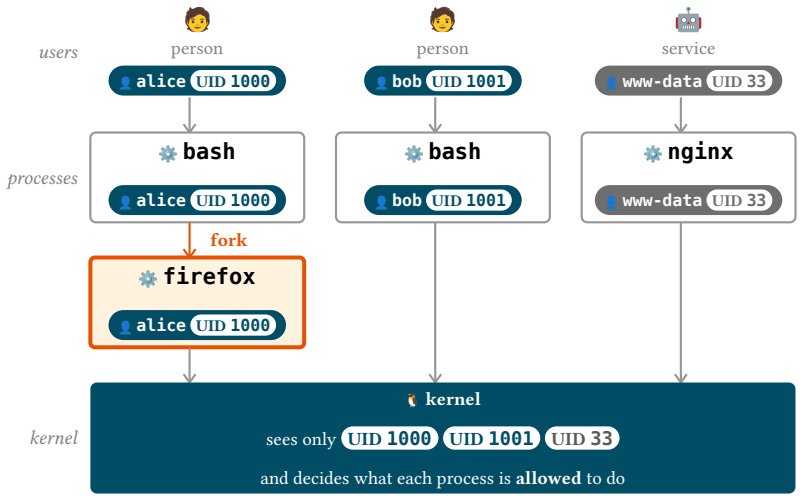


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

What Is a User?

👤 for the kernel, a user is just a **number**: the **UID**

- a user has a **name** for humans, `alice`, and a **number** for the kernel, the **UID** (*User ID*), `1000`
- not only people: each service has its own user, `www-data` is UID `33`
- every **process** carries the **UID** of its user, as a **label**, all its life
- a child **inherits** the UID: `firefox`, started by `alice`'s `bash`, also has UID `1000`
- the kernel ignores names, it checks the **UID** to decide what the process is **allowed** to do



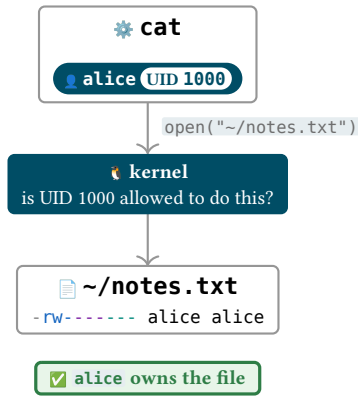
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A User Is a Number

🏷️ the kernel checks the label of a process for **every action**

```
$ cat ~/notes.txt
buy milk
```

📄 the process of `cat` runs under `alice`, the file belongs to `alice`



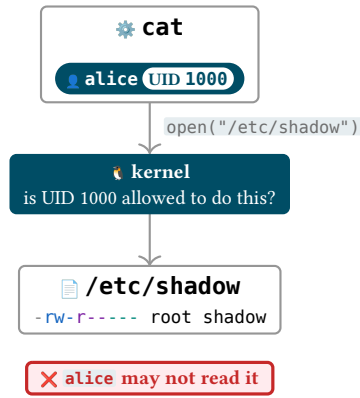
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A User Is a Number

🏷️ the kernel checks the label of a process for **every action**

```
$ cat ~/notes.txt
buy milk
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

🔒 `/etc/shadow` can be read only by `root` and the group `shadow`



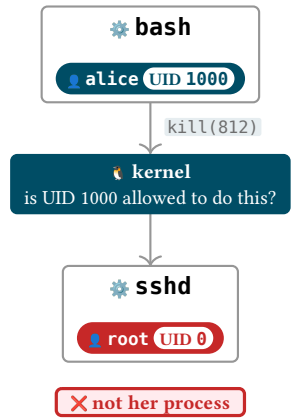
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A User Is a Number

🏷️ the kernel checks the label of a process for **every action**

```
$ cat ~/notes.txt
buy milk
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
$ kill 812
bash: kill: (812) - Operation not permitted
```

⚙️ not only files: signals, ports below 1024, the clock, mounting disks … are all checked



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Properties of a User

what the system knows about `alice`

Property	Example
username	`alice`
UID - user ID	`1000`
full name	`Alice Smith`
primary group (**GID**)	`1000`
home directory	`/home/alice`
shell	`/bin/bash`
password	*a hash*, not the password

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Properties of a User

🪪 what the system knows about `alice`

Property	Example
username	`alice`
UID - user ID	`1000`
full name	`Alice Smith`
primary group (**GID**)	`1000`
home directory	`/home/alice`
shell	`/bin/bash`
password	*a hash*, not the password

🔢 the UID is what matters

the **kernel** only knows the **UID**
the username is for *humans*, it is translated to the UID by programs

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Properties of a User

🪪 what the system knows about `alice`

Property	Example
username	`alice`
UID - user ID	`1000`
full name	`Alice Smith`
primary group (**GID**)	`1000`
home directory	`/home/alice`
shell	`/bin/bash`
password	*a hash*, not the password

🔢 the UID is what matters

the **kernel** only knows the **UID**
the username is for *humans*, it is translated to the UID by programs

⚠️ two names, one UID

two usernames with the **same UID** are the **same user** for the kernel

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

UID Ranges

🔢 most distributions split the UIDs like this

UID	Kind	Used for	Examples
`0`	👑 `root`	the administrator, **no restrictions**	`root`
`1 - 999`	⚙️ system users	services, they **cannot log in**	`daemon`, `www-data`, `sshd`
`1000 -`	🧑 regular users	people	`alice`, `bob`
`65534`	🚫 `nobody`	a user that owns **nothing**	`nobody`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

UID Ranges

🔢 most distributions split the UIDs like this

UID	Kind	Used for	Examples
`0`	👑 `root`	the administrator, **no restrictions**	`root`
`1 - 999`	⚙️ system users	services, they **cannot log in**	`daemon`, `www-data`, `sshd`
`1000 -`	🧑 regular users	people	`alice`, `bob`
`65534`	🚫 `nobody`	a user that owns **nothing**	`nobody`

> 💡 **a system user isolates a service**
>
> if `nginx` is hacked, the attacker is `www-data`, not `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

UID Ranges

🔢 most distributions split the UIDs like this

UID	Kind	Used for	Examples
`0`	👑 `root`	the administrator, **no restrictions**	`root`
`1 - 999`	⚙️ system users	services, they **cannot log in**	`daemon`, `www-data`, `sshd`
`1000 -`	🧑 regular users	people	`alice`, `bob`
`65534`	🚫 `nobody`	a user that owns **nothing**	`nobody`

> 💡 **a system user isolates a service**
>
> if `nginx` is hacked, the attacker is `www-data`, not `root`

the limits are set in `/etc/login.defs` (`UID_MIN`, `SYS_UID_MAX`, ...)

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Users Database

`/etc/passwd` *and how programs read it*

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.3.1 - *The /etc/passwd File*
     - Section 7.3.2 - *Special Users*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Local Users Database

📄 plain text files in `/etc`, one line for every user

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
bob:x:1001:1001:Bob Jones,,,:/home/bob:/bin/bash
```

`/etc/passwd` (shortened)

📒 **`/etc/passwd`**

- one **user** on every line
- the fields are separated by `:`
- 🌍 **everyone** can read it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Local Users Database

📄 plain text files in `/etc`, one line for every user

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
bob:x:1001:1001:Bob Jones,,,:/home/bob:/bin/bash
```

`/etc/passwd` (shortened)

👑 **`root`**

always the **first line**, UID `0`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Local Users Database

📄 plain text files in `/etc`, one line for every user

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
bob:x:1001:1001:Bob Jones,,,:/home/bob:/bin/bash
```

`/etc/passwd` (shortened)

⚙️ system users

UID below `1000`, created when a service is installed

the shell `/usr/sbin/nologin` refuses to log in

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Local Users Database

📄 plain text files in `/etc`, one line for every user

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
bob:x:1001:1001:Bob Jones,,,:/home/bob:/bin/bash
```

`/etc/passwd` (shortened)

🧑 regular users

UID from `1000`, a home directory in `/home` and a real shell

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Local Users Database

📄 plain text files in `/etc`, one line for every user

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
bob:x:1001:1001:Bob Jones,,,:/home/bob:/bin/bash
```

`/etc/passwd` (shortened)

> 💡 **the name has nothing special**
>
> *passwd* comes from the time when the **password** was kept in this file

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/passwd

🔍 the fields of one line

```
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

📒 **7 fields, separated by `:`**

```
$ grep alice /etc/passwd
```

shows the line of `alice`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/passwd

🔍 the fields of one line

alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash

🏷️ field 1 **username**

the name used to log in: lowercase letters, digits, `-` and `_`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/passwd

🔍 the fields of one line

```
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

🔒 field 2 **password**

`x` - the password is kept in `/etc/shadow`
in the past, the **hash** of the password was here

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/passwd

🔍 the fields of one line

```
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

🔢 field 3 **UID**

the user ID, the **only** thing the kernel knows

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/passwd

🔍 the fields of one line

```
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

👥 field 4 **GID**

the ID of the **primary group** of the user

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/passwd

🔍 the fields of one line

```
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

🪪 field 5 **GECOS**

the full name, then the room, the phone numbers … separated by `,`
the name comes from an old operating system, *General Electric Comprehensive OS*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/passwd

🔍 the fields of one line

```
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

🏠 field 6 **home directory**

the current directory after login, `~` and `$HOME`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/passwd

🔍 the fields of one line

```
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

🐚 field 7 **shell**

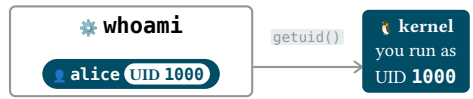
the program started after login, `/usr/sbin/nologin` refuses the login

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

From UID to Name

📚 every program that shows a user name reads `/etc/passwd`, through the C library

- the kernel gives programs only **UIDs**: `getuid()` tells `whoami` that it runs as UID `1000`

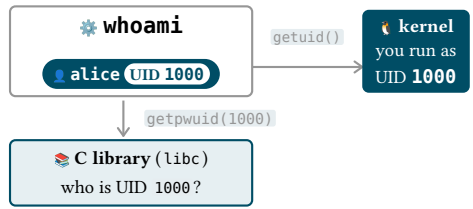


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

From UID to Name

📚 every program that shows a user name reads `/etc/passwd`, through the C library

- the kernel gives programs only **UIDs**: `getuid()` tells `whoami` that it runs as UID `1000`
- to find the **name**, the program calls a function of the C library:
  - `getpwuid(1000)` - the user with this UID
  - `getpwnam("alice")` - the user with this name
  - `getpwent()` - all the users, one by one

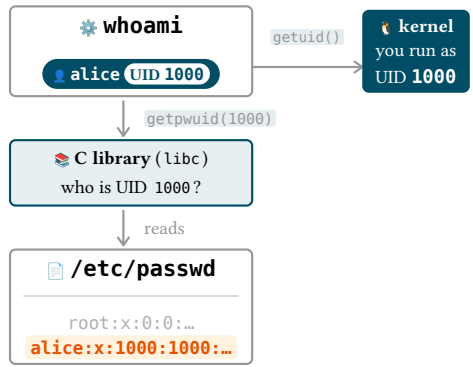


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

From UID to Name

📚 every program that shows a user name reads `/etc/passwd`, through the C library

- the kernel gives programs only **UIDs**: `getuid()` tells `whoami` that it runs as UID `1000`
- to find the **name**, the program calls a function of the C library:
  - `getpwuid(1000)` - the user with this UID
  - `getpwnam("alice")` - the user with this name
  - `getpwent()` - all the users, one by one
- `libc` reads `/etc/passwd` and returns a `struct passwd`: name, UID, GID, home, shell

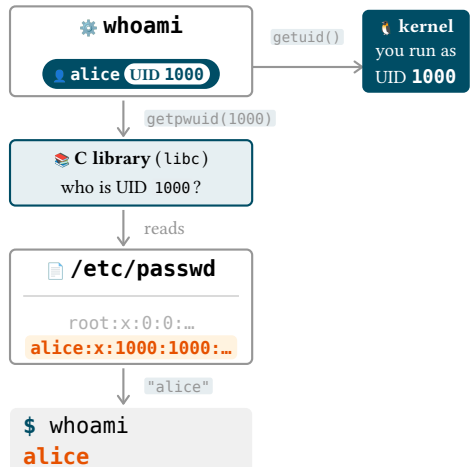


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

From UID to Name

📚 every program that shows a user name reads `/etc/passwd`, through the C library

- the kernel gives programs only **UIDs**: `getuid()` tells `whoami` that it runs as UID `1000`
- to find the **name**, the program calls a function of the C library:
  - `getpwuid(1000)` - the user with this UID
  - `getpwnam("alice")` - the user with this name
  - `getpwent()` - all the users, one by one
- `libc` reads `/etc/passwd` and returns a `struct passwd`: name, UID, GID, home, shell
- `ls`, `ps`, `id`, `whoami`, the prompt ... all do this, so `/etc/passwd` must be readable by **everyone**



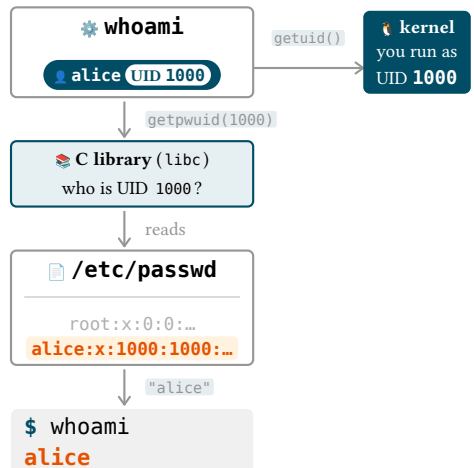
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

From UID to Name

📚 every program that shows a user name reads `/etc/passwd`, through the C library

- the kernel gives programs only **UIDs**: `getuid()` tells `whoami` that it runs as UID `1000`
- to find the **name**, the program calls a function of the C library:
  - `getpwuid(1000)` - the user with this UID
  - `getpwnam("alice")` - the user with this name
  - `getpwent()` - all the users, one by one
- `libc` reads `/etc/passwd` and returns a `struct passwd`: name, UID, GID, home, shell
- `ls`, `ps`, `id`, `whoami`, the prompt ... all do this, so `/etc/passwd` must be readable by **everyone**

> ⚠️ **never read `/etc/passwd` directly (`grep`, `cut`): the users may also come from other places**



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`getent`

🔎 *get entries*: finds a user, wherever it comes from

```
$ getent passwd alice
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

Command	Finds
`getent passwd alice`	the user with this **name**
`getent passwd 1000`	the user with this **UID**
`getent passwd`	**all** the users

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`getent`

🔎 *get entries*: finds a user, wherever it comes from

```
$ getent passwd alice
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
$ getent passwd 1000
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
```

Command	Finds
`getent passwd alice`	the user with this **name**
`getent passwd 1000`	the user with this **UID**
`getent passwd`	**all** the users

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`getent`

🔎 *get entries*: finds a user, wherever it comes from

```
$ getent passwd alice
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
$ getent passwd 1000
alice:x:1000:1000:Alice Smith,,,:/home/alice:/bin/bash
$ getent passwd
root:x:0:0:root:/root:/bin/bash
...
bob:x:1001:1001:Bob Jones,,,:/home/bob:/bin/bash
```

Command	Finds
`getent passwd alice`	the user with this **name**
`getent passwd 1000`	the user with this **UID**
`getent passwd`	**all** the users

✅ **the same format as `/etc/passwd`: use `getent` in scripts, instead of `grep` or `cut`**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Local Password Storage

hashes, salt and `/etc/shadow`

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.3.3 - *The /etc/shadow File*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Passwords Database

🔐 `/etc/shadow`, one line for every user of `/etc/passwd`

```
root:!:20737:0:99999:7:::
daemon:*:20737:0:99999:7:::
www-data:*:20737:0:99999:7:::
nobody:*:20737:0:99999:7:::
alice:$y$j9T$Xr3a…$9mQe…:20737:0:99999:7:::
bob:$y$j9T$Lp0w…$Tz7k…:20737:0:99999:7:::
```

`/etc/shadow` (shortened)

🔐 **`/etc/shadow`**

- the same users as `/etc/passwd`, in the same order
- the **hash** of the password and when it expires
- the fields are separated by `:`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Passwords Database

🔐 `/etc/shadow`, one line for every user of `/etc/passwd`

```
root:!:20737:0:99999:7:::
daemon:*:20737:0:99999:7:::
www-data:*:20737:0:99999:7:::
nobody:*:20737:0:99999:7:::
alice:$y$j9T$Xr3a…$9mQe…:20737:0:99999:7:::
bob:$y$j9T$Lp0w…$Tz7k…:20737:0:99999:7:::
```

`/etc/shadow` (shortened)

👑 `root`

`!` - **locked**: no password works, use `sudo` instead

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Passwords Database

🔐 `/etc/shadow`, one line for every user of `/etc/passwd`

```
root:!:20737:0:99999:7:::
daemon:*:20737:0:99999:7:::
www-data:*:20737:0:99999:7:::
nobody:*:20737:0:99999:7:::
alice:$y$j9T$Xr3a…$9mQe…:20737:0:99999:7:::
bob:$y$j9T$Lp0w…$Tz7k…:20737:0:99999:7:::
```

`/etc/shadow` (shortened)

> ⚙️ **system users**
>
> `*` - they **never** had a password, nobody can log in as them

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Passwords Database

🔐 `/etc/shadow`, one line for every user of `/etc/passwd`

```
root:!:20737:0:99999:7:::
daemon:*:20737:0:99999:7:::
www-data:*:20737:0:99999:7:::
nobody:*:20737:0:99999:7:::
alice:$y$j9T$Xr3a…$9mQe…:20737:0:99999:7:::
bob:$y$j9T$Lp0w…$Tz7k…:20737:0:99999:7:::
```

`/etc/shadow` (shortened)

🧑 regular users

the **hash** of the password, never the password itself

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Passwords Database

🔐 `/etc/shadow`, one line for every user of `/etc/passwd`

```
root:!:20737:0:99999:7:::
daemon:*:20737:0:99999:7:::
www-data:*:20737:0:99999:7:::
nobody:*:20737:0:99999:7:::
alice:$y$j9T$Xr3a…$9mQe…:20737:0:99999:7:::
bob:$y$j9T$Lp0w…$Tz7k…:20737:0:99999:7:::
```

`/etc/shadow` (shortened)

```
$ getent shadow alice   # nothing, alice may not read it
```

> 🔒 **only `root` can read it**
>
> unlike `/etc/passwd`, that everyone reads

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:  :  :
```

> 🔐 **9 fields, separated by `:`**
>
> the line of `alice`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

alice:yj9T$Xr3…$9mQ…:20737:0:99999:7:::

🏷️ field 1 **username**

the same name as in `/etc/passwd`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:  :  :
```

🔢 field 2 **password hash**

`$algorithm$salt$hash`, `$y$` is *yescrypt*, `$6$` is *SHA-512*

`!` or `*` - the account is **locked**, no password can log in

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:  :  :
```

📅 field 3 **last change**

the day of the last password change, counted from 1 January 1970: `20737` is 11 October 2026

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:  :  :
```

⏳ field 4 **minimum age**

days before the password can be changed again

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:  :  :
```

⌛ field 5 **maximum age**

days after which the password **must** be changed, `99999` - never

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:  :  :
```

⚠ field 6 **warning**

days before the expiry when the user is warned

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:□:  :
```

💤 field 7 **inactive**

days after the expiry when the account is still usable, empty - no limit

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:  :▢:
```

🚫 field 8 **expire date**

the day when the account is disabled, empty - never

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/shadow

🔍 the fields of one line

```
alice:$y$j9T$Xr3…$9mQ…:20737:0:99999:7:  :  :
```



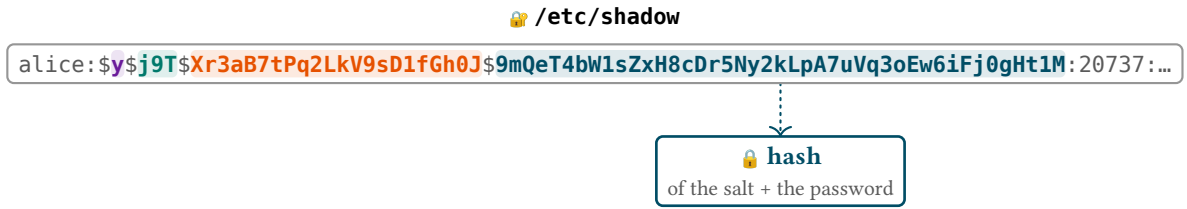
🧩 field 9 **reserved**

not used

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Storing a Password

- a hash, with salt, in `/etc/shadow`



- the password is never stored, only its **hash**: a **one way** function, it cannot be turned back into the password

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Storing a Password

🧂 a hash, with salt, in `/etc/shadow`



- the password is never stored, only its **hash**: a **one way** function, it cannot be turned back into the password
- a random **salt** is added to the password before hashing: two users with the same password get **different** hashes

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Storing a Password

a hash, with salt, in `/etc/shadow`



- the password is never stored, only its **hash**: a **one way** function, it cannot be turned back into the password
- a random **salt** is added to the password before hashing: two users with the same password get **different** hashes
- the **algorithm** and its **parameters** are stored too: they are needed to check the password

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Checking a Password

🔍 the hash cannot be reversed, so the password is hashed again and compared



1️⃣ `login` reads the line of `alice` from `/etc/shadow`: the **algorithm**, the **parameters**, the **salt** and the **stored hash**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Checking a Password

🔍 the hash cannot be reversed, so the password is hashed again and compared

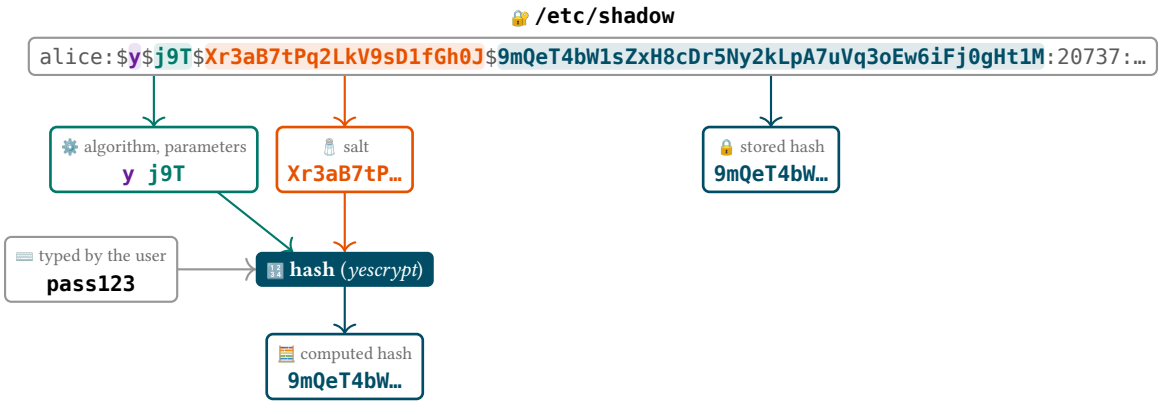


2️⃣ the typed password is hashed with the same **algorithm**, **parameters** and **salt**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Checking a Password

🔍 the hash cannot be reversed, so the password is hashed again and compared

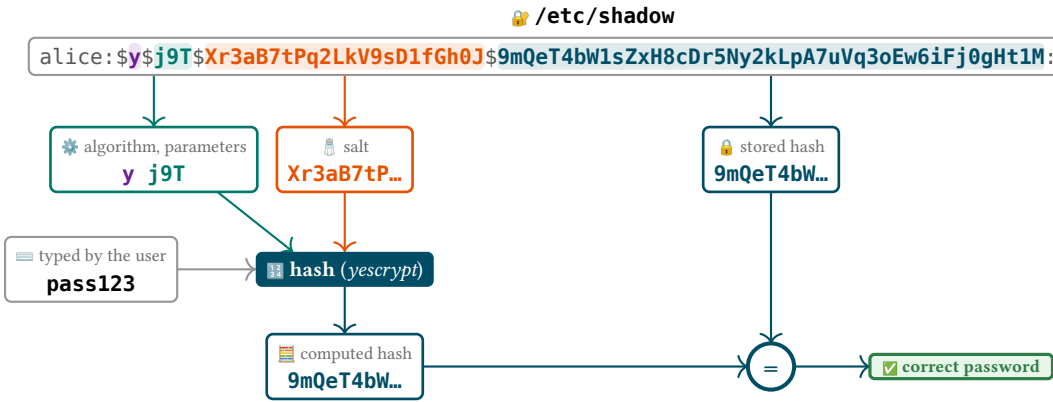


3️⃣ the result is a new hash, computed now

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Checking a Password

🔍 the hash cannot be reversed, so the password is hashed again and compared

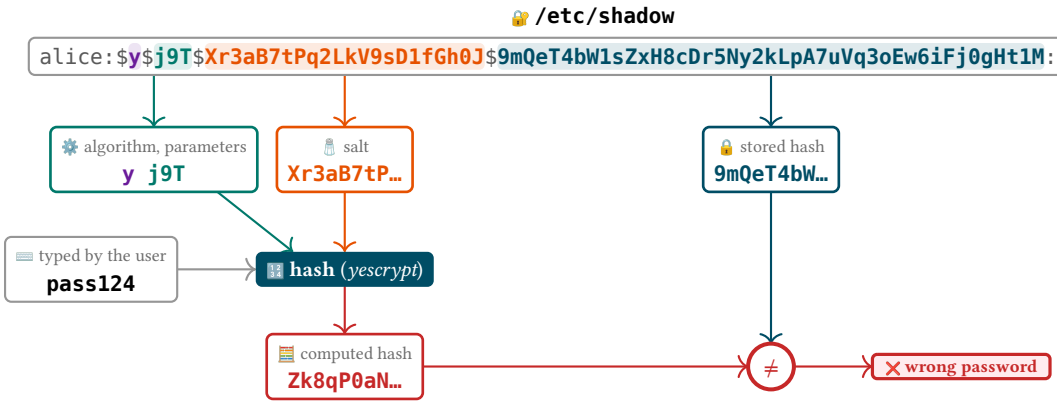


4️⃣ the same as the **stored hash**: the password is correct, without ever storing it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Checking a Password

🔍 the hash cannot be reversed, so the password is hashed again and compared



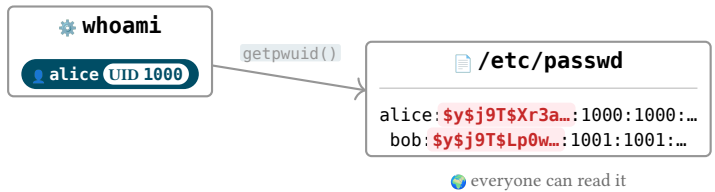
5️⃣ one wrong letter gives a completely different hash: the password is wrong

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Why `/etc/shadow`?

🔐 the hashes cannot stay in a file that everyone reads

- `/etc/passwd` must be readable by **everyone**, for `getpwuid()`

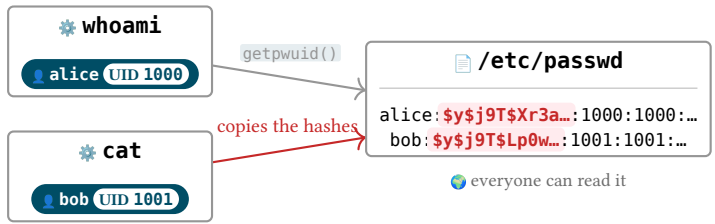


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Why `/etc/shadow`?

🔐 the hashes cannot stay in a file that everyone reads

- `/etc/passwd` must be readable by **everyone**, for `getpwuid()`
- in the past, the **hash** was in its second field, so anyone could copy it

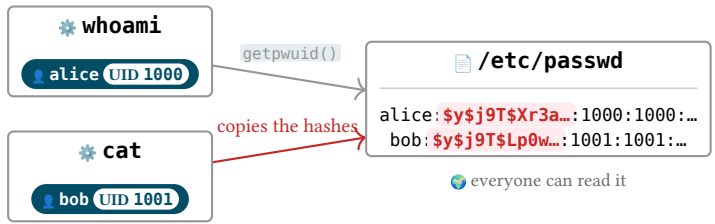


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Why `/etc/shadow`?

🔐 the hashes cannot stay in a file that everyone reads

- `/etc/passwd` must be readable by **everyone**, for `getpwuid()`
- in the past, the **hash** was in its second field, so anyone could copy it
- the password check works for an attacker too: guess, hash, compare, **offline**



🔨 **an offline attack, on `bob`'s own computer**

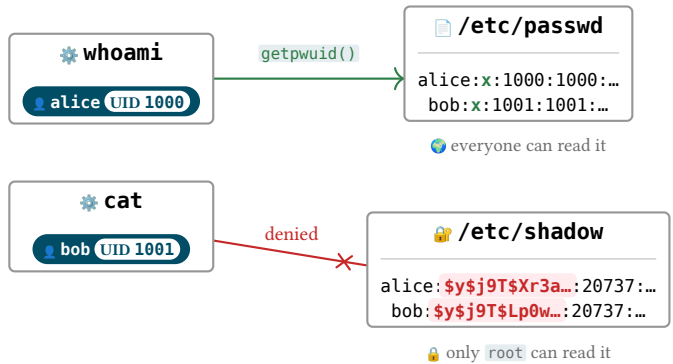
hash(salt + "123456") = the stolen hash? ❌
hash(salt + "password") = the stolen hash? ❌
hash(salt + "pass123") = the stolen hash? ✅
millions of tries per second: no log, no delay, no one sees it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Why `/etc/shadow`?

🔐 the hashes cannot stay in a file that everyone reads

- `/etc/passwd` must be readable by **everyone**, for `getpwuid()`
- in the past, the **hash** was in its second field, so anyone could copy it
- the password check works for an attacker too: guess, hash, compare, **offline**
- ✅ the hashes moved to `/etc/shadow`, `/etc/passwd` keeps only an `x`

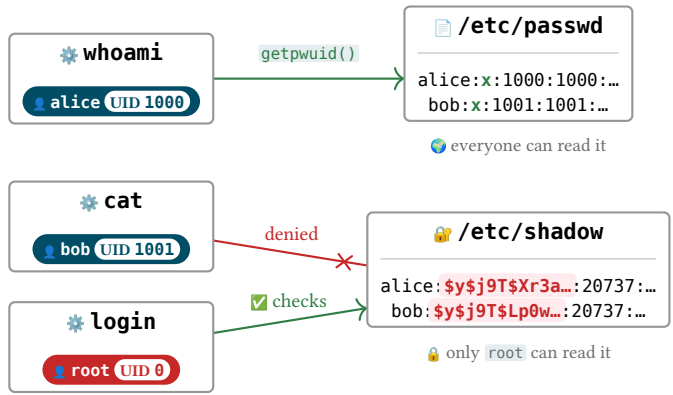


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Why `/etc/shadow`?

🔐 the hashes cannot stay in a file that everyone reads

- `/etc/passwd` must be readable by **everyone**, for `getpwuid()`
- in the past, the **hash** was in its second field, so anyone could copy it
- the password check works for an attacker too: guess, hash, compare, **offline**
- ✅ the hashes moved to `/etc/shadow`, `/etc/passwd` keeps only an `x`
- only `root` reads it: `login`, `sudo` and `passwd` run as `root` to check the password



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Groups

rights for several users at once

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.3.5 - *Working with Groups*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Groups and Users

👥 a group is a **set of users** that share the same rights

- every group has a **name** and a **GID**

Group	GID	👤 alice	👤 bob	👤 carol
alice	1000			
bob	1001			
carol	1002			
students	1100			
sudo	27			
docker	998			

⭐ primary group ● supplementary group

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Groups and Users

👥 a group is a **set of users** that share the same rights

- every group has a **name** and a **GID**
- every user has **one primary group** (the GID in `/etc/passwd`), usually a group with the same name as the user, only for them

Group	GID	👤 alice	👤 bob	👤 carol
alice	1000	⭐		
bob	1001		⭐	
carol	1002			⭐
students	1100			
sudo	27			
docker	998			

⭐ primary group ● supplementary group

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Groups and Users

👥 a group is a **set of users** that share the same rights

- every group has a **name** and a **GID**
- every user has **one primary group** (the GID in `/etc/passwd`), usually a group with the same name as the user, only for them
- and any number of **supplementary groups**

Group	GID	👤 alice	👤 bob	👤 carol
alice	1000	⭐		
bob	1001		⭐	
carol	1002			⭐
students	1100	●	●	
sudo	27	●		
docker	998			●

⭐ primary group ● supplementary group

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/group

📒 the local groups database

```
students:x:1100:alice,bob
```

📒 **4 fields, separated by `:`**

```
$ grep students /etc/group   # only the local file
$ getent group students      # the groups database
```

🌍 everyone can read it, `getent` asks the network too, like `getent passwd`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/group

📒 the local groups database

students:x:1100:alice,bob

🏷️ field 1 **group name**

the name of the group

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/group

📒 the local groups database

```
students:x:1100:alice,bob
```

🔒 field 2 **password**

`x` - almost never used

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/group

📒 the local groups database

```
students:x:1100:alice,bob
```

🔢 field 3 **GID**

the group ID, the kernel uses the GID

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

/etc/group

📒 the local groups database

```
students:x:1100:alice,bob
```

👥 field 4 **members**

the users that have this group as a **supplementary** group, separated by `,`

⚠️ the users that have it as **primary** group are not listed here

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Logging In

how a process gets the label of a user

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.4 - *getty and login*
     - Section 7.9.3 - *User Identification, Authentication, and Authorization*
     - Section 7.9.4 - *Using Libraries for User Information*
2. `man 5 nsswitch.conf` - where the C library looks for the users
3. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 9 - *Pornirea sistemului*
     - Section 9.6.1 - *Pornirea terminalelor de login*

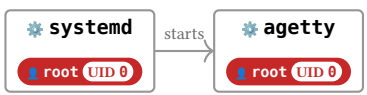
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Logging In Locally

🖥️ a text terminal, `/dev/tty1`

```
[PID 1]  systemd            user: root
└── [PID 20]  agetty tty1   user: root
        # prints "login:" and reads the name
```

> ⌨️ **the name**
>
> `agetty` waits on the terminal, as `root`, for a user name



> 💡 **a graphical login (`gdm`, `sddm`) and `ssh` do the same steps**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

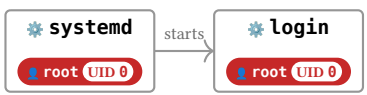
Logging In Locally

🖥️ a text terminal, `/dev/tty1`

```
[PID 1]  systemd            user: root
└── [PID 20]  login alice   user: root
        # agetty became login (exec)
        # reads the password
```

🔑 the password

`agetty` **becomes** `login` (`exec`), still `root`, it asks for the password



💡 a graphical login (`gdm`, `sddm`) and `ssh` do the same steps

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

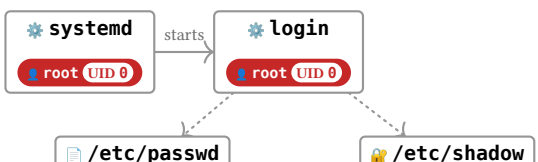
Logging In Locally

🖥️ a text terminal, `/dev/tty1`

```
[PID 1]  systemd            user: root
└── [PID 20]  login alice   user: root
        # is the password right?
        #   /etc/passwd  - alice, UID 1000
        #   /etc/shadow  - the hash
```

✅ the check

`login` finds `alice`[1], hashes the password with the salt from `/etc/shadow` and compares



💡 **a graphical login (`gdm`, `sddm`) and `ssh` do the same steps**

[1] `login` asks the C library (`getpwnam()`, `getspnam()`), that may also find the users on a network server

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Logging In Locally

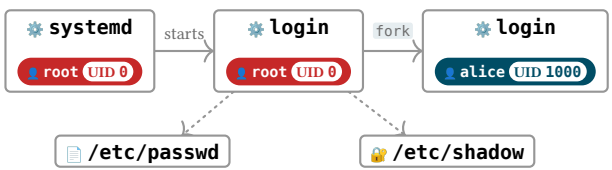
🖥️ a text terminal, `/dev/tty1`

```
[PID 1]  systemd            user: root
└── [PID 20]  login alice   user: root
      │   # fork
      └── [PID 30]  login   user: alice
            # setgid(1000), initgroups()
            # setuid(1000)
```

🏷️ the label

a child of `login` sets the **GID**, the **groups** and the **UID** of `alice`

⚠️ after `setuid` it can **never** go back to `root`



💡 **a graphical login (`gdm`, `sddm`) and `ssh` do the same steps**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Logging In Locally

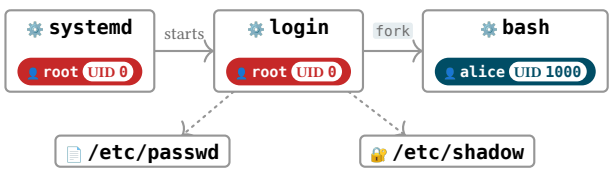
🖥️ a text terminal, `/dev/tty1`

```
[PID 1]  systemd            user: root
└── [PID 20]  login alice   user: root
      └── [PID 30]  bash    user: alice
              # exec /bin/bash in /home/alice
              # HOME, USER, SHELL are set
```

🐚 **the shell**

it changes to the home directory and runs the **shell** from `/etc/passwd`

every program `alice` starts inherits the label

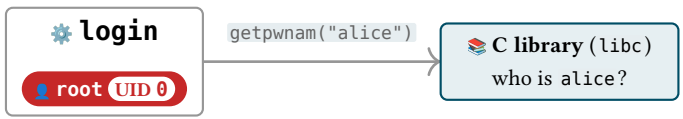


💡 **a graphical login (`gdm`, `sddm`) and `ssh` do the same steps**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Where Are the Users?

🔎 `login` asks the C library, the C library knows where to look

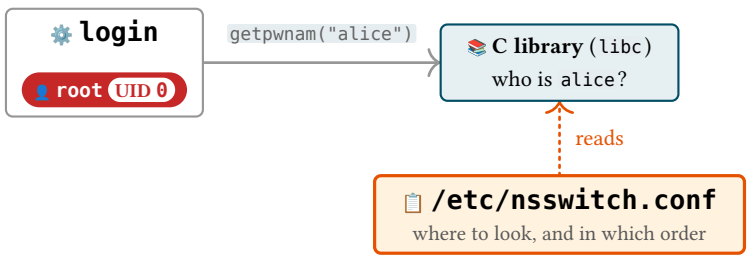


⚙️ `login` does not open the files: it asks the **C library** for the user

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Where Are the Users?

🔎 `login` asks the C library, the C library knows where to look

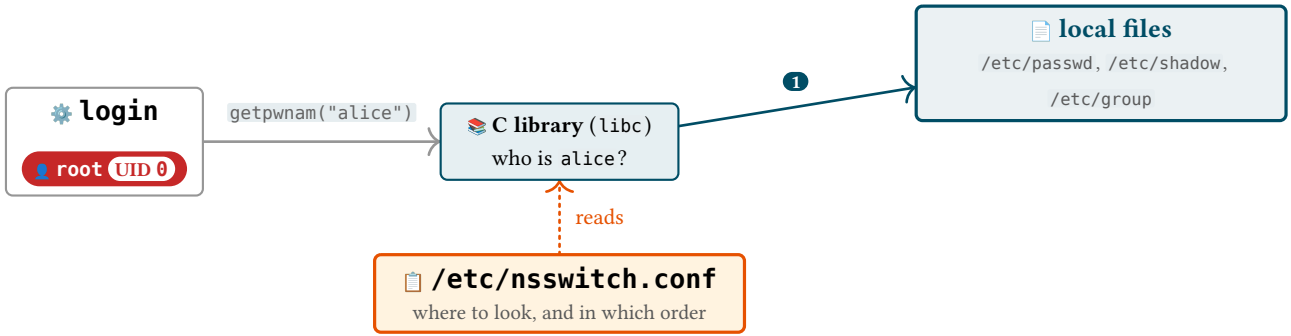


📋 the C library reads `/etc/nsswitch.conf`: the **places** where users are kept

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Where Are the Users?

🔎 `login` asks the C library, the C library knows where to look

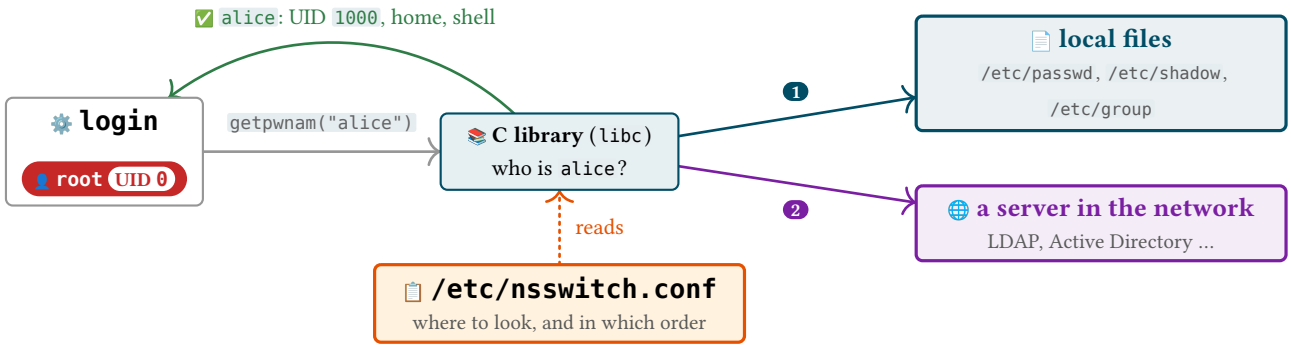


📄 usually the **local files** first: `/etc/passwd`, `/etc/shadow`, `/etc/group`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Where Are the Users?

🔎 `login` asks the C library, the C library knows where to look



🌐 then, in a university or a company, a **server** in the network that keeps the accounts

> 💡 **`id`, `ls -l`, `getent`, `sudo` ... all ask the same C library: they all see the same users**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The `root` User

the administrator, UID 0

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.9.1 - *User IDs and User Switching*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The `root` User

👑 the user with the UID `0`

- the kernel **skips** most of the checks for the UID `0`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The `root` User

👑 the user with the UID `0`

- the kernel **skips** most of the checks for the UID `0`
- reads and writes **any** file, sends signals to **any** process

```
root@laptop:~# cat /etc/shadow
root:!:20737:0:99999:7:::
...
root@laptop:~# kill 812
root@laptop:~# rm -rf /   # never do this
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The `root` User

👑 the user with the UID `0`

- the kernel **skips** most of the checks for the UID `0`
- reads and writes **any** file, sends signals to **any** process
- installs software, adds users, mounts disks, changes the network

```
root@laptop:~# cat /etc/shadow
root:!:20737:0:99999:7:::
...
root@laptop:~# kill 812
root@laptop:~# rm -rf /   # never do this
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The `root` User

👑 the user with the UID `0`

- the kernel **skips** most of the checks for the UID `0`
- reads and writes **any** file, sends signals to **any** process
- installs software, adds users, mounts disks, changes the network
- its home directory is `/root`, not `/home/root`

```
root@laptop:~# cat /etc/shadow
root:!:20737:0:99999:7:::
...
root@laptop:~# kill 812
root@laptop:~# rm -rf /   # never do this
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The `root` User

👑 the user with the UID `0`

- the kernel **skips** most of the checks for the UID `0`
- reads and writes **any** file, sends signals to **any** process
- installs software, adds users, mounts disks, changes the network
- its home directory is `/root`, not `/home/root`
- ⚠️ a mistake as `root` can **destroy** the whole system

```
root@laptop:~# cat /etc/shadow
root:!:20737:0:99999:7:::
...
root@laptop:~# kill 812
root@laptop:~# rm -rf /   # never do this
```

⚠️ work as a regular user

become `root` only when a command **needs** it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`root` Is Just a Name

🏷️ the kernel checks the **UID**, not the name

```
root:x:0:0:root:/root:/bin/bash

alice:x:1000:1000:…:/home/alice:/bin/bash
```

`/etc/passwd`

- `root` is only the **name** of the UID `0`, the first field of `/etc/passwd`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`root` Is Just a Name

🏷️ the kernel checks the **UID**, not the name

```
root admin:x:0:0:root:/root:/bin/bash

alice:x:1000:1000:…:/home/alice:/bin/bash
```

`/etc/passwd`

- it can be **renamed**: change the name in `/etc/passwd` and in `/etc/shadow`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`root` Is Just a Name

🏷️ the kernel checks the **UID**, not the name

```
admin:x:0:0:root:/root:/bin/bash
toor:x:0:0:Bourne-again Superuser:/root:/bin/sh
alice:x:1000:1000:…:/home/alice:/bin/bash
```

/etc/passwd

- `toor` - a second name for UID `0`, used on some BSD systems

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`root` Is Just a Name

🏷️ the kernel checks the **UID**, not the name

```
admin:x:0:0:root:/root:/bin/bash
toor:x:0:0:Bourne-again Superuser:/root:/bin/sh
alice:x:1000:1000:…:/home/alice:/bin/bash
```

`/etc/passwd`

- two names, the **same user**: the kernel sees only `0`

> ⚠️ **do not rename `root`**
>
> many programs and scripts expect the name `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Becoming `root`

`sudo` *and* `su`

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 2 - *Basic Commands and Directory Hierarchy*
     - Section 2.20 - *Running Commands as the Superuser*
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.9.1 - *User IDs and User Switching*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*

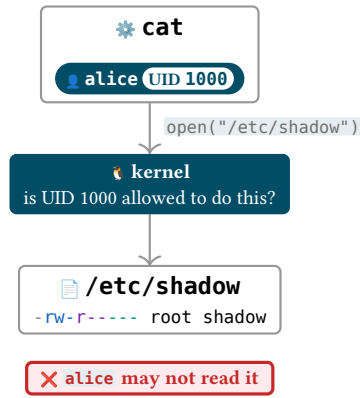
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`sudo` Changes the Label

👑 the same command, under another UID

```
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

🔒 `cat` runs under `alice`, UID `1000`: `/etc/shadow` can be read only by `root` and the group `shadow`



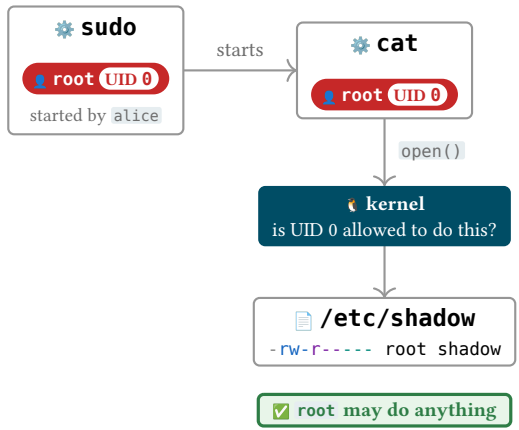
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`sudo` Changes the Label

👑 the same command, under another UID

```
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
$ sudo cat /etc/shadow
root:*:20737:0:99999:7:::
```

👑 `sudo` runs as `root`, UID `0`, and starts `cat` under the same user: the kernel allows everything to UID `0`



 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`sudo` and `su`

🔑 two ways to run commands as `root`

🛡️ `sudo` - run one command as `root`

```
$ sudo dnf install htop
[sudo] password for alice:
```

- asks for **your** password
- only the users allowed in `/etc/sudoers`
- every command is written in a log

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`sudo` and `su`

🔑 two ways to run commands as `root`

🛡️ `sudo` - run one command as `root`

```
$ sudo dnf install htop
[sudo] password for alice:
```

- asks for **your** password
- only the users allowed in `/etc/sudoers`
- every command is written in a log

🔄 `su` - switch user, start a shell

```
$ su -
Password:
```

- asks for the password of **`root`**
- everyone who uses it must know that password
- nothing tells **who** became `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`sudo` and `su`

🔑 two ways to run commands as `root`

🛡️ `sudo` - run one command as `root`

```
$ sudo dnf install htop
[sudo] password for alice:
```

- asks for **your** password
- only the users allowed in `/etc/sudoers`
- every command is written in a log

🔄 `su` - switch user, start a shell

```
$ su -
Password:
```

- asks for the password of **`root`**
- everyone who uses it must know that password
- nothing tells **who** became `root`

💡 `root` may have no password

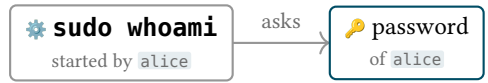
on some systems (Ubuntu, macOS) it is `!` in `/etc/shadow`, only `sudo` works

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who can use `sudo`

🛡️ a program that decides who may become `root`

```
alice@laptop:~$ sudo whoami
[sudo] password for alice:
```



🔑 `sudo` asks for the password of **`alice`**, not the one of `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who can use `sudo`

🛡️ a program that decides who may become `root`

```
alice@laptop:~$ sudo whoami
[sudo] password for alice:
root
```



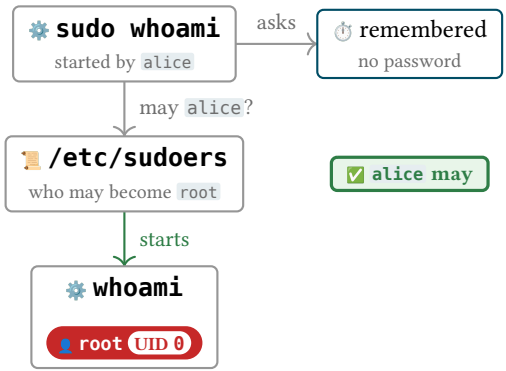
📜 `/etc/sudoers` says who may become `root`: `alice` may, so `whoami` runs as `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who can use `sudo`

🛡️ a program that decides who may become `root`

```
alice@laptop:~$ sudo whoami
[sudo] password for alice:
root
alice@laptop:~$ sudo whoami
root
```



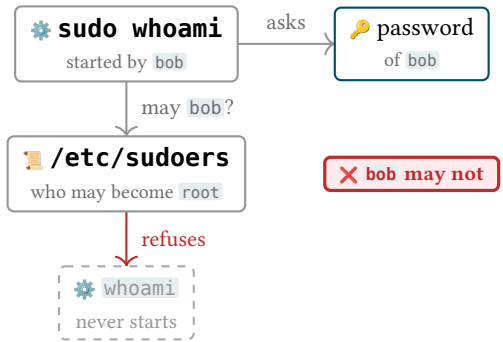
⏱️ no password again for the next 5 minutes, in the same terminal

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who can use `sudo`

🛡️ a program that decides who may become `root`

```
alice@laptop:~$ sudo whoami
[sudo] password for alice:
root
alice@laptop:~$ sudo whoami
root
bob@laptop:~$ sudo whoami
[sudo] password for bob:
bob is not in the sudoers file.
```



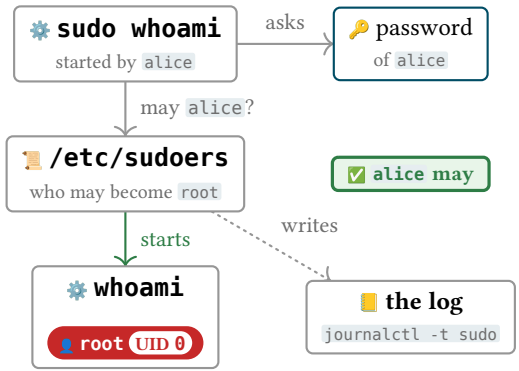
🚫 `bob` is not in `/etc/sudoers`: `sudo` refuses, `whoami` never starts

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who can use `sudo`

🛡️ a program that decides who may become `root`

```
alice@laptop:~$ sudo whoami
[sudo] password for alice:
root
alice@laptop:~$ sudo whoami
root
bob@laptop:~$ sudo whoami
[sudo] password for bob:
bob is not in the sudoers file.
alice@laptop:~$ sudo journalctl -t sudo
alice : TTY=pts/0 ; USER=root ; COMMAND=/usr/bin/whoami
alice : TTY=pts/0 ; USER=root ; COMMAND=/usr/bin/whoami
bob : user NOT in sudoers ; TTY=pts/1 ; USER=root ; ...
```



📒 every attempt is written in the log: **who** ran **what**, **as whom**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Using `sudo`

🧰 the options you will need

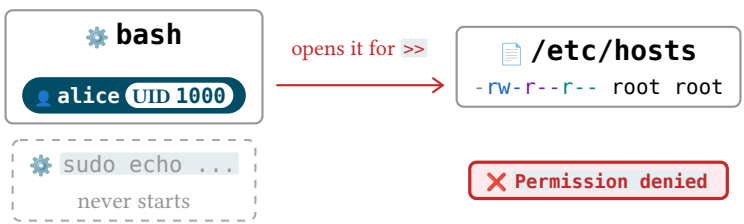
Command	What it does
`sudo command`	runs `command` as `root`
`sudo -u bob command`	runs `command` as `bob`
`sudo -E command`	keeps **your** environment variables (no `env_reset`), e.g. `http_proxy`
`sudo -i`	starts a **login shell** of `root`: `root`'s home and variables
`sudo -s`	starts a shell as `root`, in the current directory
`sudo -l`	lists the commands you may run
`sudo -k`	forgets the password now, the next `sudo` asks again
`sudo !!`	runs the **previous** command again, with `sudo` (a `bash` trick)

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Redirect Trap

🪤 `sudo` runs the **command**, not the whole line

```
$ sudo echo "192.168.1.20 printer" >> /etc/hosts
bash: /etc/hosts: Permission denied
```



💡 **the shell of `alice` opens the file for `>>`, before `sudo` starts: the redirect is not part of the command**

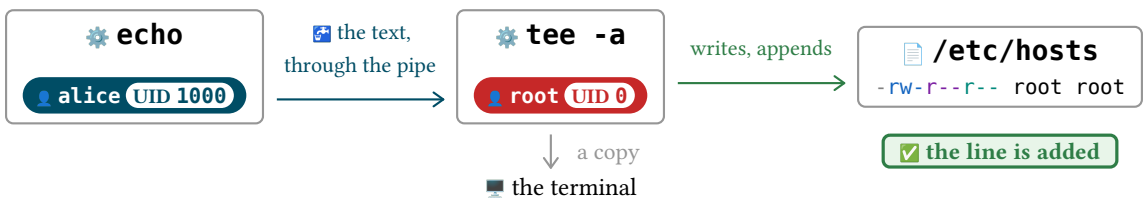
 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Redirect Trap



🪤 `sudo` runs the **command**, not the whole line

```
$ sudo echo "192.168.1.20 printer" >> /etc/hosts
bash: /etc/hosts: Permission denied
$ echo "192.168.1.20 printer" | sudo tee -a /etc/hosts
192.168.1.20 printer
```



💡 **`tee` writes to the file and to the screen, as `root`; `-a` appends, like `>>`**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

File Permissions

`r`, `w` *and* `x` *for files and directories*

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 2 - *Basic Commands and Directory Hierarchy*
     - Section 2.17 - *File Modes and Permissions*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*
     - Section 5.5

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who Owns a File?

🔍 `ls -l` shows the owner, the group and the permissions

```
- rw- r-- --- 1 alice students 1234 Oct 11 09:00 notes.txt
```

📄 **every file has**

- **one owner** (a user) and **one group**
- three sets of permissions: for the **owner**, the **group** and the **others**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who Owns a File?

🔍 `ls -l` shows the owner, the group and the permissions

```
- rw- r-- --- 1 alice students 1234 Oct 11 09:00 notes.txt
```

> 👤 field 6 **owner**
>
> the user that owns the file, usually the one that created it

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who Owns a File?

🔍 `ls -l` shows the owner, the group and the permissions

```
- rw- r-- --- 1 alice students 1234 Oct 11 09:00 notes.txt
```

> 👥 field 7 **group**
>
> the group of the file, usually the primary group of its creator

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who Owns a File?

🔍 `ls -l` shows the owner, the group and the permissions

```
-rw- r-- --- 1 alice students 1234 Oct 11 09:00 notes.txt
```

👤 field 2 **`u` - user, the owner**

`rw-` - `alice` can **read** and **write**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who Owns a File?

🔍 `ls -l` shows the owner, the group and the permissions

```
- rw- r-- --- 1 alice students 1234 Oct 11 09:00 notes.txt
```

👥 field 3 **`g` - group**

`r--` - the members of `students` can only **read**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Who Owns a File?

🔍 `ls -l` shows the owner, the group and the permissions

```
- rw- r--[---] 1 alice students 1234 Oct 11 09:00 notes.txt
```

🌍 field 4 **`o` - others**

`---` - everybody else can do **nothing**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`r`, `w` and `x`

📖 the same letters, a different meaning for directories

	Value	📄 For a file	📁 For a directory
`r` - read	`4`	read the content (`cat`)	**list** the names inside (`ls`)
`w` - write	`2`	change the content	**create**, **delete**, **rename** the files inside
`x` - execute	`1`	run it as a program	**enter** it (`cd`) and reach the files inside

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`r`, `w` and `x`

📖 the same letters, a different meaning for directories

	Value	📄 For a file	📁 For a directory
`r` - read	`4`	read the content (`cat`)	**list** the names inside (`ls`)
`w` - write	`2`	change the content	**create**, **delete**, **rename** the files inside
`x` - execute	`1`	run it as a program	**enter** it (`cd`) and reach the files inside

⚠️ creating, renaming or deleting a file is writing the directory

the names of the files are kept in their directory: with `w` on a directory you can rename or delete **any** file in it, even if it is not yours

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`r`, `w` and `x`

📖 the same letters, a different meaning for directories

	Value	📄 For a file	📁 For a directory
`r` - read	`4`	read the content (`cat`)	**list** the names inside (`ls`)
`w` - write	`2`	change the content	**create**, **delete**, **rename** the files inside
`x` - execute	`1`	run it as a program	**enter** it (`cd`) and reach the files inside

⚠️ creating, renaming or deleting a file is writing the directory

the names of the files are kept in their directory: with `w` on a directory you can rename or delete **any** file in it, even if it is not yours

💡 to reach a file you need `x` on every directory of its path

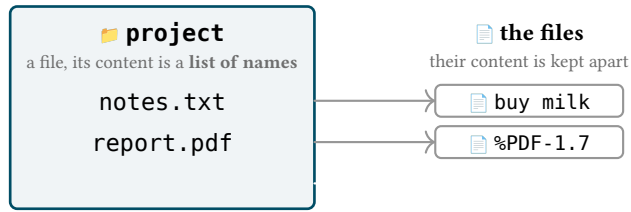
`cat /home/alice/notes.txt` needs `x` on `/`, `/home` and `/home/alice`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A Directory Is a List

📋 a file read by the OS: the list of the files inside

```
$ ls -ld project
drwxr-xr-x 2 alice alice 4096 Oct 11 09:00 project
$ ls -l project
-r--r--r-- 1 alice alice  9 Oct 11 09:00 notes.txt
-r--r--r-- 1 alice alice 58 Oct 11 09:00 report.pdf
```



📋 a directory is a file the OS reads: a **list of names**, each pointing to a file

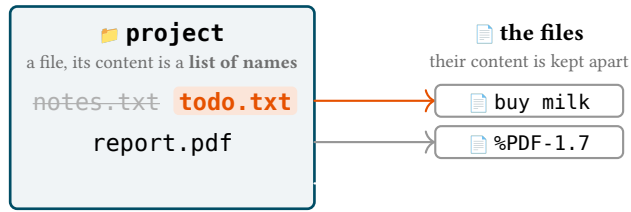
`r` **reads** this list, `w` **changes** it, `x` **uses** it to reach a file

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A Directory Is a List

📋 a file read by the OS: the list of the files inside

```
$ ls -ld project
drwxr-xr-x 2 alice alice 4096 Oct 11 09:00 project
$ ls -l project
-r--r--r-- 1 alice alice  9 Oct 11 09:00 notes.txt
-r--r--r-- 1 alice alice 58 Oct 11 09:00 report.pdf
$ mv project/notes.txt project/todo.txt
```



✏️ **rename** - `w` on the directory: only the name in the list changes, the file is not touched

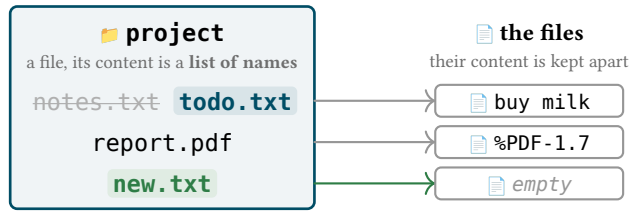
> ✅ **it is the list that changes**
>
> the files are `r--r--r--`, but `alice` has `w` on `project`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A Directory Is a List

📋 a file read by the OS: the list of the files inside

```
$ ls -ld project
drwxr-xr-x 2 alice alice 4096 Oct 11 09:00 project
$ ls -l project
-r--r--r-- 1 alice alice  9 Oct 11 09:00 notes.txt
-r--r--r-- 1 alice alice 58 Oct 11 09:00 report.pdf
$ mv project/notes.txt project/todo.txt
$ touch project/new.txt
```



➕ **create** - `w` on the directory: a new name is added to the list

> ✅ **it is the list that changes**
>
> the files are `r--r--r--`, but `alice` has `w` on `project`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

A Directory Is a List

📋 a file read by the OS: the list of the files inside

```
$ ls -ld project
drwxr-xr-x 2 alice alice 4096 Oct 11 09:00 project
$ ls -l project
-r--r--r-- 1 alice alice  9 Oct 11 09:00 notes.txt
-r--r--r-- 1 alice alice 58 Oct 11 09:00 report.pdf
$ mv project/notes.txt project/todo.txt
$ touch project/new.txt
$ rm project/report.pdf
rm: remove write-protected regular file 'project/report.pdf'? y
```



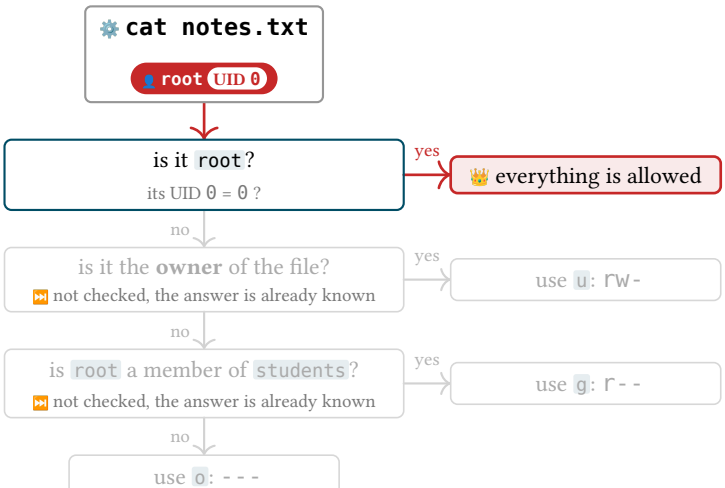
🗑️ **delete** - `w` on the directory: the name is removed from the list, even if the file is read only

> ✅ **it is the list that changes**
>
> the files are `r--r--r--`, but `alice` has `w` on `project`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Permissions Apply?

⚖️ the kernel checks **in order**, the first answer `yes` decides





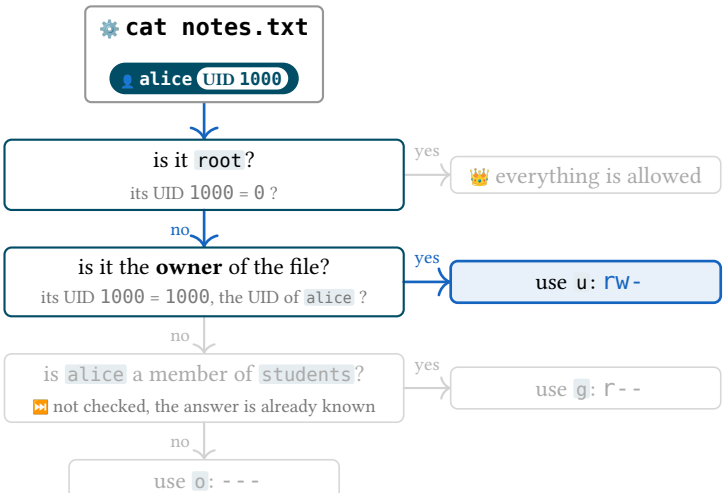
`- rw- r-- ---`

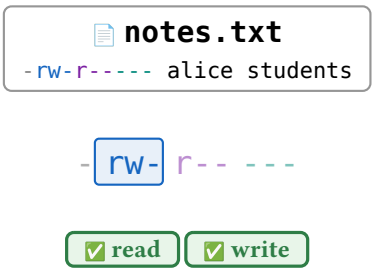
✅ read ✅ write

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Permissions Apply?

⚖️ the kernel checks **in order**, the first answer `yes` decides

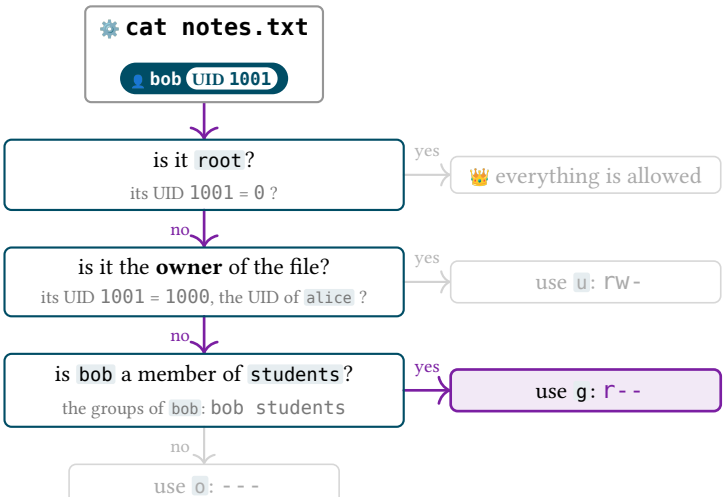


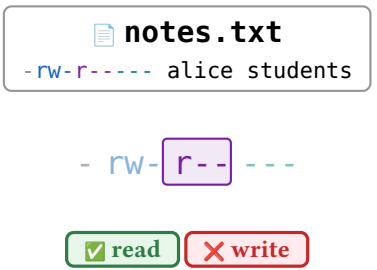


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Permissions Apply?

⚖️ the kernel checks **in order**, the first answer `yes` decides

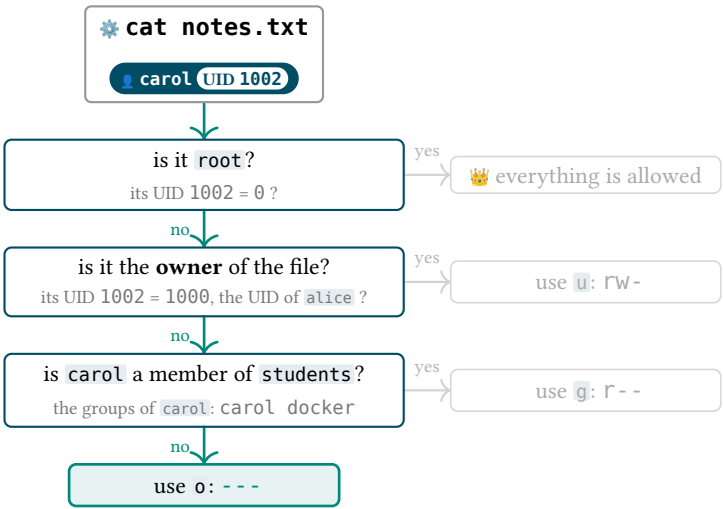


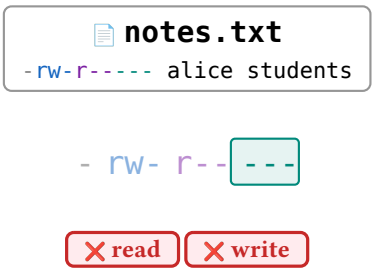


 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Permissions Apply?

⚖️ the kernel checks **in order**, the first answer `yes` decides

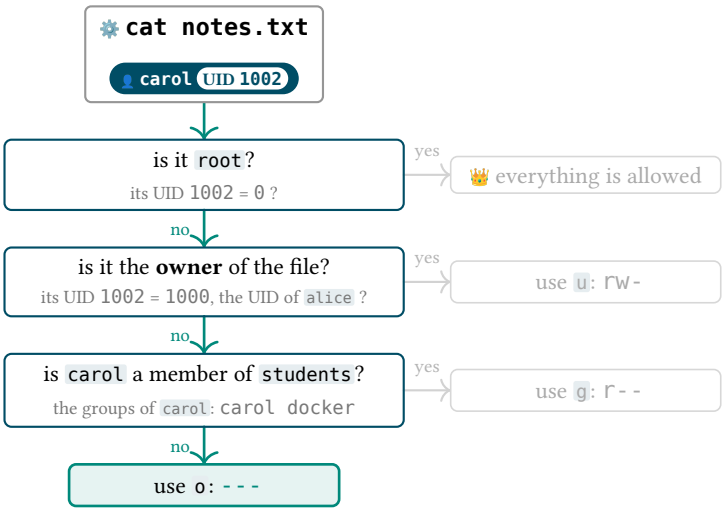




 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Permissions Apply?

⚖️ the kernel checks **in order**, the first answer `yes` decides





`- rw- r-- ---`

❌ read ❌ write

⚠️ **only one set is used**

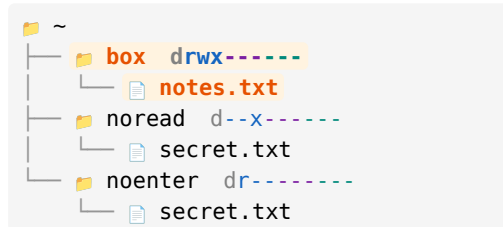
with `----rw----` the owner can do nothing, even if the group can: the owner already got `yes`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Directories in Practice

📁 `r` lists, `x` enters

```
$ ls -l
drwx------ 2 alice alice 4096 Oct 11 09:00 box
d--x------ 2 alice alice 4096 Oct 11 09:00 noread
dr-------- 2 alice alice 4096 Oct 11 09:00 noenter
```



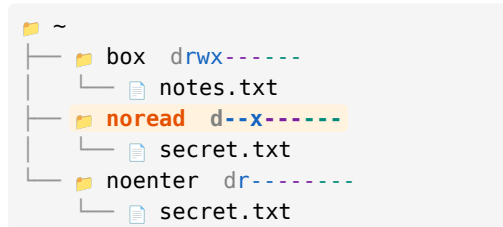
📁 three directories of `alice`, with different permissions for her

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Directories in Practice

📁 `r` lists, `x` enters

```
$ ls -l
drwx------ 2 alice alice 4096 Oct 11 09:00 box
d--x------ 2 alice alice 4096 Oct 11 09:00 noread
dr-------- 2 alice alice 4096 Oct 11 09:00 noenter
$ ls noread
ls: cannot open directory 'noread': Permission denied
```



🙈 `noread` has no `r`: the **names** cannot be listed

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Directories in Practice



📁 `r` lists, `x` enters

```
$ ls -l
drwx------ 2 alice alice 4096 Oct 11 09:00 box
d--x------ 2 alice alice 4096 Oct 11 09:00 noread
dr-------- 2 alice alice 4096 Oct 11 09:00 noenter
$ ls noread
ls: cannot open directory 'noread': Permission denied
$ cat noread/secret.txt
hello
```

```
📁 ~
├── 📁 box  drwx------
│   └── 📄 notes.txt
├── 📁 noread  d--x------
│   └── 📄 secret.txt
└── 📁 noenter  dr--------
    └── 📄 secret.txt
```

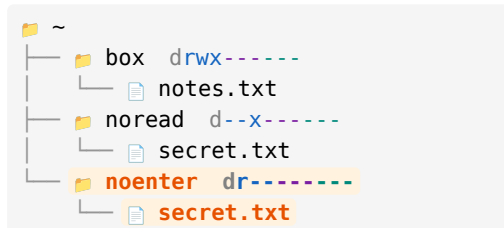
🔑 but it has `x`: a file whose name you **know** can still be opened

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Directories in Practice

📁 `r` lists, `x` enters

```
$ ls -l
drwx------ 2 alice alice 4096 Oct 11 09:00 box
d--x------ 2 alice alice 4096 Oct 11 09:00 noread
dr-------- 2 alice alice 4096 Oct 11 09:00 noenter
$ ls noread
ls: cannot open directory 'noread': Permission denied
$ cat noread/secret.txt
hello
$ ls noenter
ls: cannot access 'noenter/secret.txt': Permission denied
secret.txt
```



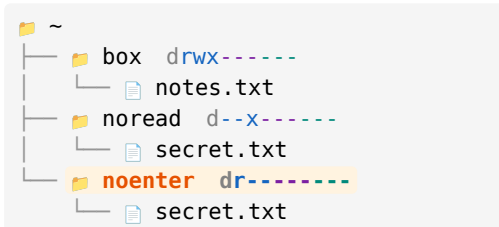
👀 `noenter` has `r` without `x`: the names are listed, but nothing more

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Directories in Practice

📁 `r` lists, `x` enters

```
$ ls -l
drwx------ 2 alice alice 4096 Oct 11 09:00 box
d--x------ 2 alice alice 4096 Oct 11 09:00 noread
dr-------- 2 alice alice 4096 Oct 11 09:00 noenter
$ ls noread
ls: cannot open directory 'noread': Permission denied
$ cat noread/secret.txt
hello
$ ls noenter
ls: cannot access 'noenter/secret.txt': Permission denied
secret.txt
$ cd noenter
bash: cd: noenter: Permission denied
```



🚪 without `x`: the directory cannot be **entered**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

chmod

✏️ change the permissions, only the owner (or `root`) can do it

🔤 with letters

who `u` `g` `o` `a` (all)

how `+` add, `-` remove, `=` set

what `r` `w` `x`

```
$ chmod u+x script.sh
$ chmod g+w,o-r notes.txt
$ chmod a=r readme.txt
$ chmod -R g+rX project/
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

chmod

✏️ change the permissions, only the owner (or `root`) can do it

🔤 with letters

who `u` `g` `o` `a` (all)

how `+` add, `-` remove, `=` set

what `r` `w` `x`

```
$ chmod u+x script.sh
$ chmod g+w,o-r notes.txt
$ chmod a=r readme.txt
$ chmod -R g+rX project/
```

🔢 with numbers (octal)

`r` = 4, `w` = 2, `x` = 1, add them for `u`, `g`, `o`

rw-	r--	---
4+2 = **6**	4 = **4**	**0**

```
$ chmod 640 notes.txt
$ chmod 755 script.sh
```

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

umask

🎭 the permissions that new files **do not** get

```
$ umask
0022
```

🎭 a mask, not a permission

`umask` lists the bits that are **removed** from every new file

`022` - remove `w` for the group and for the others

🧬 inherited

every process has a `umask`, inherited from its parent, usually `022` (`002` on some systems)

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`umask`

🎭 the permissions that new files **do not** get

```
$ umask
0022
$ touch notes.txt
$ ls -l notes.txt
-rw-r--r-- 1 alice alice 0 Oct 11 09:00 notes.txt
```

📄 **a new file**

the program asks for	`rw- rw- rw-`	666
− `umask` removes	`--- -w- -w-`	022
the result	**`rw- r-- r--`**	**644**

📄 **a new file is never executable**

programs create files with `rw-rw-rw-` (`666`), the `umask` only **removes** bits

Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

umask



🎭 the permissions that new files **do not** get

```
$ umask
0022
$ touch notes.txt
$ ls -l notes.txt
-rw-r--r-- 1 alice alice 0 Oct 11 09:00 notes.txt
$ mkdir project
$ ls -ld project
drwxr-xr-x 2 alice alice 4096 Oct 11 09:00 project
```

📁 a new directory		
the program asks for	`rwx rwx rwx`	777
– `umask` removes	`--- -w- -w-`	022
the result	`rwx r-x r-x`	**755**

📁 a new directory gets `x`

directories are created with `rwxrwxrwx` (777): they need `x` to be entered

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

umask

🎭 the permissions that new files **do not** get

```
$ umask
0022
$ touch notes.txt
$ ls -l notes.txt
-rw-r--r-- 1 alice alice 0 Oct 11 09:00 notes.txt
$ mkdir project
$ ls -ld project
drwxr-xr-x 2 alice alice 4096 Oct 11 09:00 project
$ umask 077
$ touch secret.txt
$ ls -l secret.txt
-rw------- 1 alice alice 0 Oct 11 09:00 secret.txt
```

📄 **with `umask 077`**

the program asks for	rw- rw- rw-	666
− `umask` removes	--- rwx rwx	077
the result	rw- --- ---	**600**

🔒 private by default

`umask 077`: nobody else can read the new files, put it in `~/.bashrc` to keep it

Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`chown` and `chgrp`

🔄 change the owner and the group

```
$ sudo chown bob notes.txt
```

- the owner becomes `bob`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`chown` and `chgrp`

🔄 change the owner and the group

```
$ sudo chown bob notes.txt
$ sudo chown bob:students notes.txt
```

- the owner **and** the group, `user:group`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`chown` and `chgrp`

🔄 change the owner and the group

```
$ sudo chown bob notes.txt
$ sudo chown bob:students notes.txt
$ chgrp students notes.txt
```

- only the group, the owner may do it, if they are a member of the new group

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`chown` and `chgrp`

🔄 change the owner and the group

```
$ sudo chown bob notes.txt
$ sudo chown bob:students notes.txt
$ chgrp students notes.txt
$ chown carol notes.txt
chown: changing ownership of 'notes.txt': Operation
not permitted
```

- only `root` can give a file to another user

> 💡 **why only `root`?**
>
> otherwise, anyone could fill the disk quota of another user, or hide what they did

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Permissions

setuid, setgid and the sticky bit

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 2 - *Basic Commands and Directory Hierarchy*
     - Section 2.17 - *File Modes and Permissions*
2. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*
     - Section 5.5

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Permissions

✨ three more bits, shown in place of an `x`

	`ls -l` shows	the rights are	📄 On a file	📁 On a directory
🪄 **setuid**	`-rwsr-xr-x` `/usr/bin/passwd`	`-rwxr-xr-x` + **setuid**	runs as the **owner** of the file, not as the user who starts it	-

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Permissions

✨ three more bits, shown in place of an `x`

	`ls -l` shows	the rights are	📄 On a file	📁 On a directory
🪄 setuid	`-rwsr-xr-x` `/usr/bin/passwd`	`-rwxr-xr-x` + setuid	runs as the **owner** of the file, not as the user who starts it	-
👥 setgid	`drwxrwsr-x` `/srv/project`	`drwxrwxr-x` + setgid	runs with the **group** of the file	the new files get the **group** of the directory, not of their creator

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Permissions

✨ three more bits, shown in place of an `x`

	`ls -l` shows	the rights are	📄 On a file	📁 On a directory
🪄 **setuid**	`-rwsr-xr-x` `/usr/bin/passwd`	`-rwxr-xr-x` + **setuid**	runs as the **owner** of the file, not as the user who starts it	-
👥 **setgid**	`drwxrwsr-x` `/srv/project`	`drwxrwxr-x` + **setgid**	runs with the **group** of the file	the new files get the **group** of the directory, not of their creator
📌 **sticky**	`drwxrwxrwt` `/tmp`	`drwxrwxrwx` + **sticky**	-	in a directory everyone can write, you can delete or rename **only your own** files

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Special Permissions

✨ three more bits, shown in place of an `x`

	`ls -l` shows	the rights are	📄 On a file	📁 On a directory
🪄 **setuid**	`-rwsr-xr-x` `/usr/bin/passwd`	`-rwxr-xr-x` + **setuid**	runs as the **owner** of the file, not as the user who starts it	-
👥 **setgid**	`drwxrwsr-x` `/srv/project`	`drwxrwxr-x` + **setgid**	runs with the **group** of the file	the new files get the **group** of the directory, not of their creator
📌 **sticky**	`drwxrwxrwt` `/tmp`	`drwxrwxrwx` + **sticky**	-	in a directory everyone can write, you can delete or rename **only your own** files

💡 **a capital `S` or `T`: the bit is set, but the `x` under it is not**

`-rwSr--r--` means `-rw-r--r--` + **setuid** `drwxrwxrwT` means `drwxrwxrw-` + **sticky**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`setuid` at Work

🪄 the `s` bit (*setuid*): the program runs as its **owner**, not as the user who starts it

```
alice@laptop:~$
```



🐚 `alice` has a shell, `bash`, that runs as her

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`setuid` at Work

🪄 the `s` bit (*setuid*): the program runs as its **owner**, not as the user who starts it

```
alice@laptop:~$ passwd
```



⌨️ she types `passwd`: `bash` reads the program file `/usr/bin/passwd`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`setuid` at Work

🪄 the `s` bit (*setuid*): the program runs as its **owner**, not as the user who starts it

```
alice@laptop:~$ passwd
```



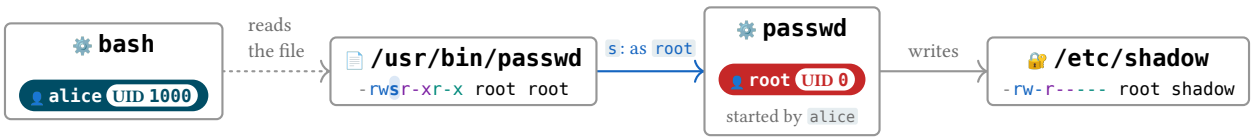
🪄 the file belongs to `root` and has the `s` bit: the new process runs as `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`setuid` at Work

🪄 the `s` bit (*setuid*): the program runs as its **owner**, not as the user who starts it

```
alice@laptop:~$ passwd
```



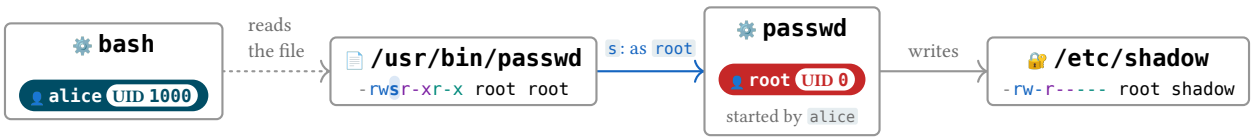
✍️ as `root`, `passwd` may write her new hash in `/etc/shadow`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`setuid` at Work

🪄 the `s` bit (*setuid*): the program runs as its **owner**, not as the user who starts it

```
alice@laptop:~$ passwd
```




```
alice@laptop:~$
```



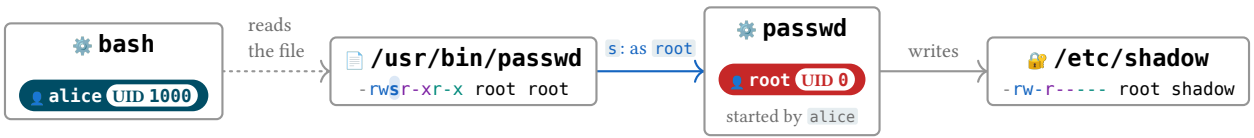
🐚 the same shell of `alice`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`setuid` at Work

🪄 the `s` bit (*setuid*): the program runs as its **owner**, not as the user who starts it

```
alice@laptop:~$ passwd
```




```
alice@laptop:~$ sudo cat /etc/shadow
```



⌨️ she types `sudo cat /etc/shadow`: `bash` reads `/usr/bin/sudo`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

`setuid` at Work

🪄 the `s` bit (*setuid*): the program runs as its **owner**, not as the user who starts it

```
alice@laptop:~$ passwd
```




```
alice@laptop:~$ sudo cat /etc/shadow
```



🛡️ `/usr/bin/sudo` is setuid `root` too: `sudo` runs as `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

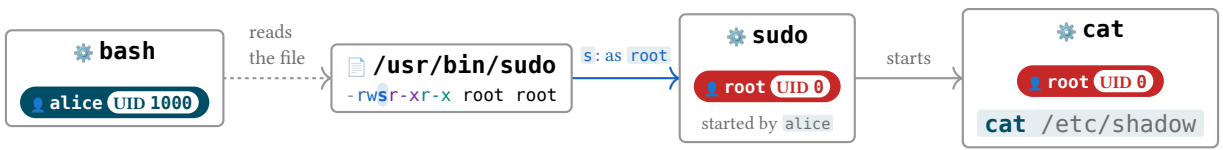
`setuid` at Work

🪄 the `s` bit (*setuid*): the program runs as its **owner**, not as the user who starts it

```
alice@laptop:~$ passwd
```




```
alice@laptop:~$ sudo cat /etc/shadow
```



✅ `sudo` checks `/etc/sudoers` and `alice`'s password, then starts `cat /etc/shadow` as `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Do Not Use `+s`

☠️ a setuid `root` program is **very dangerous**

☠️ never add `+s` to a program of `root` ☠️

it runs as its owner, `root`, for **everyone**: one bug, and **anyone** becomes `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Do Not Use `+s`

☠️ a setuid `root` program is **very dangerous**

☠️ never add `+s` to a program of `root` ☠️

it runs as its owner, `root`, for **everyone**: one bug, and **anyone** becomes `root`

☠️ `-rwsr-xr-x root root /usr/bin/bash`

a setuid `bash`: every user gets a `root` shell

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Do Not Use `+s`

☠️ a setuid `root` program is **very dangerous**

☠️ never add `+s` to a program of `root` ☠️

it runs as its owner, `root`, for **everyone**: one bug, and **anyone** becomes `root`

☠️ `-rwsr-xr-x root root /usr/bin/bash`

a setuid `bash`: every user gets a `root` shell

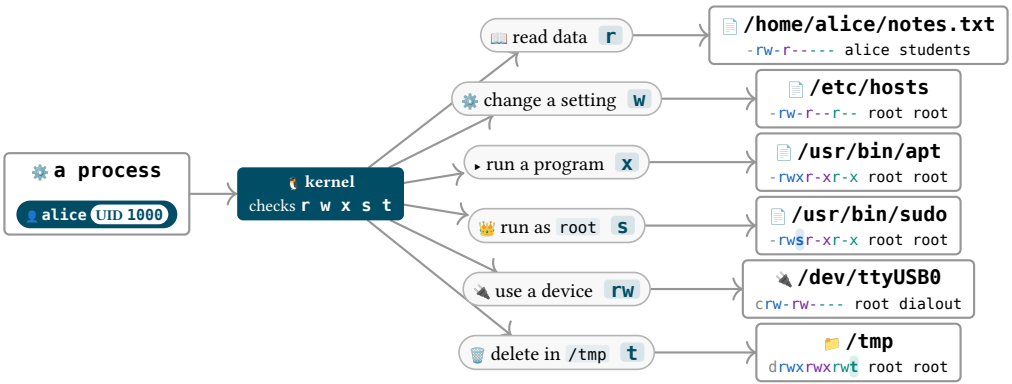
✅ **leave the `s` bit to the programs made for it: `sudo`, `passwd`, `su`**

on **your** program it is risky too: whoever runs it acts as **you**, with your files

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Everything Is a File

🗂️ data, settings, programs and devices: every right is a right on a file



> 💡 **a plain Linux system has no other privileges**
>
> every right is one of `r` `w` `x` `s` `t` on a file
>
> 👑 only `root` skips the checks

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Label of a Process

the credentials that the kernel checks

Bibliography

for this section

1. **Brian Ward**, *How LINUX Works*, 3rd Edition, No Starch Press, 2021
   - Chapter 7 - *System Configuration: Logging, System Time, Batch Jobs, and Users*
     - Section 7.9.1 - *User IDs and User Switching*
     - Section 7.9.2 - *Process Ownership, Effective UID, Real UID, and Saved UID*
2. `man 7 credentials` - the credentials of a process
3. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Credentials of a Process

🪪 an ID card that the **kernel** keeps for every process

- **real** UID - the user who **started** the process



🙋 **real**: *who you are* - `whoami` and `id` read the UIDs, the programs use the real one to know who started them

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Credentials of a Process

🪪 an ID card that the **kernel** keeps for every process

- **real** UID - the user who **started** the process
- **effective** UID - the one the kernel **checks**



🔍 **effective**: *what you may do* - the kernel compares it with the owner of a file

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Credentials of a Process

🪪 an ID card that the **kernel** keeps for every process

- **real** UID - the user who **started** the process
- **effective** UID - the one the kernel **checks**
- **saved** UID - a spare copy of the effective one



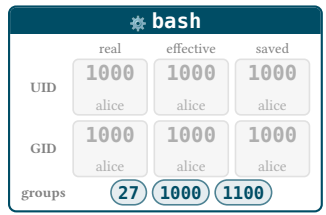
💾 **saved**: lets a program put its rights aside for a while and take them back later

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Credentials of a Process

🪪 an ID card that the **kernel** keeps for every process

- **real** UID - the user who **started** the process
- **effective** UID - the one the kernel **checks**
- **saved** UID - a spare copy of the effective one
- the same three for the **GID**, and the **supplementary groups**



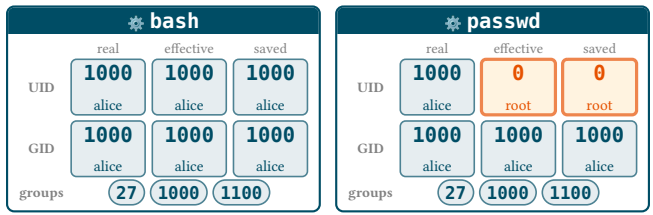
👥 the groups of `alice`: `sudo` (27), `alice` (1000) and `students` (1100); a 4th UID and GID, for the *file system*, always follow the effective ones

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Credentials of a Process

🪪 an ID card that the **kernel** keeps for every process

- **real** UID - the user who **started** the process
- **effective** UID - the one the kernel **checks**
- **saved** UID - a spare copy of the effective one
- the same three for the **GID**, and the **supplementary groups**
- usually **all the same**, but not always



🪄 `passwd` has the `setuid` bit: its effective and saved UID are the **owner** of the file, `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Seeing the Credentials



👀 `ps` and `/proc` show the ID card of any process

```
$ ps -o pid,ruser,euser,suser,comm -p $$
    PID RUSER    EUSER    SUSER    COMMAND
   4242 alice    alice    alice    bash
```

- `ruser`, `euser`, `suser` - the **real**, **effective** and **saved** user

💡 `$$` is the PID of the shell itself: all three UIDs are `alice`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Seeing the Credentials



👀 `ps` and `/proc` show the ID card of any process

```
$ ps -o pid,ruser,euser,suser,comm -p $$
    PID RUSER    EUSER    SUSER    COMMAND
   4242 alice    alice    alice    bash
$ grep -E '^(Uid|Gid|Groups):' /proc/$$/status
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
Groups: 27 1000 1100
```

- `ruser`, `euser`, `suser` - the **real**, **effective** and **saved** user
- `/proc/<PID>/status`: `Uid:` and `Gid:` are real, effective, saved, file system
- `Groups:` - the supplementary groups

💡 everyone may read the `status` file of any process; the 4th number is the *file system* UID or GID

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Seeing the Credentials



👀 `ps` and `/proc` show the ID card of any process

```
$ ps -o pid,ruser,euser,suser,comm -p $$
    PID RUSER    EUSER    SUSER    COMMAND
   4242 alice    alice    alice    bash
$ grep -E '^(Uid|Gid|Groups):' /proc/$$/status
Uid:    1000    1000    1000    1000
Gid:    1000    1000    1000    1000
Groups: 27 1000 1100
$ ps -o pid,ruser,euser,suser,comm -C passwd
    PID RUSER    EUSER    SUSER    COMMAND
   4250 alice    root     root     passwd
```

- `ruser`, `euser`, `suser` - the **real**, **effective** and **saved** user
- `/proc/<PID>/status`: `Uid:` and `Gid:` are real, effective, saved, file system
- `Groups:` - the supplementary groups
- `-C passwd` - the processes of the command `passwd`, started by `alice` in another terminal

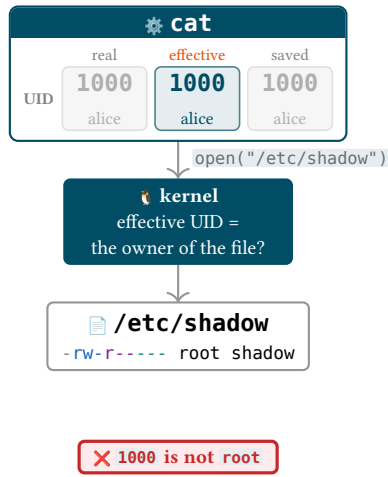
💡 `passwd` runs as `root` (*effective*), but the kernel still knows that `alice` started it (*real*)

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Credentials Are Checked

🔍 it depends on what the process asks for

- 📄 **open a file** - the *effective* UID and GIDs, against the owner and group of the file



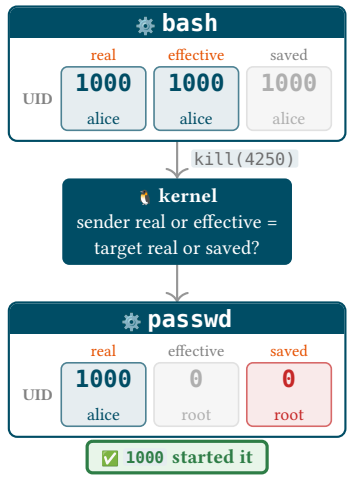
💡 `cat` runs as `alice`, `/etc/shadow` belongs to `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Credentials Are Checked

🔍 it depends on what the process asks for

- 📄 **open a file** - the *effective* UID and GIDs, against the owner and group of the file
- 📡 **send a signal** - the *real* or *effective* UID of the sender, against the *real* or *saved* UID of the target



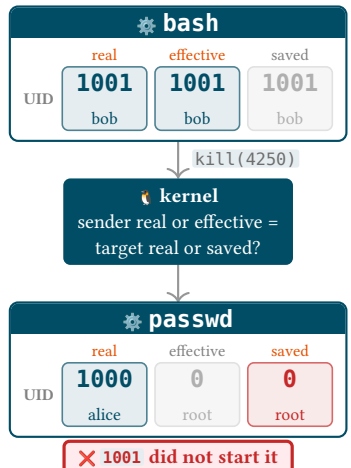
💡 `alice` may stop the `passwd` she started, even if it runs as `root`: its *real* UID is hers

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Credentials Are Checked

🔍 it depends on what the process asks for

- 📄 **open a file** - the *effective* UID and GIDs, against the owner and group of the file
- 📡 **send a signal** - the *real* or *effective* UID of the sender, against the *real* or *saved* UID of the target



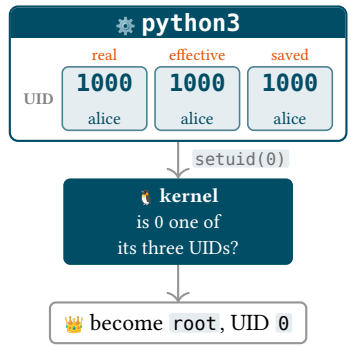
💡 `bob` may not: none of his UIDs is a UID of that `passwd`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Which Credentials Are Checked

🔍 it depends on what the process asks for

- 📄 **open a file** - the *effective* UID and GIDs, against the owner and group of the file
- 📡 **send a signal** - the *real* or *effective* UID of the sender, against the *real* or *saved* UID of the target
- 🔄 **change its own UIDs** - only to a UID it **already has**
- 👑 only an effective UID `0` may take **any** UID



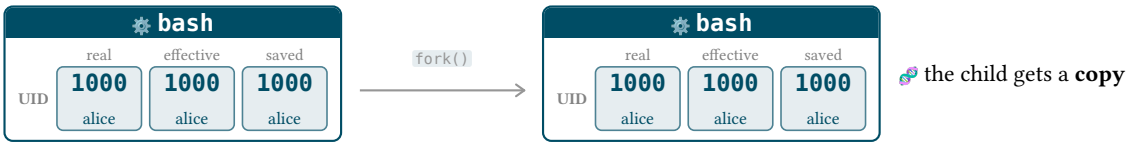
❌ it is not

💡 a process **cannot** make itself `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Label Changes

🧬 a process gets its label from its parent

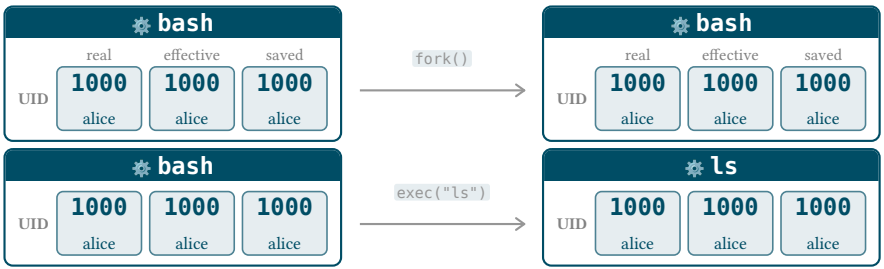


🧬 the child gets a **copy**

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Label Changes

🧬 a process gets its label from its parent



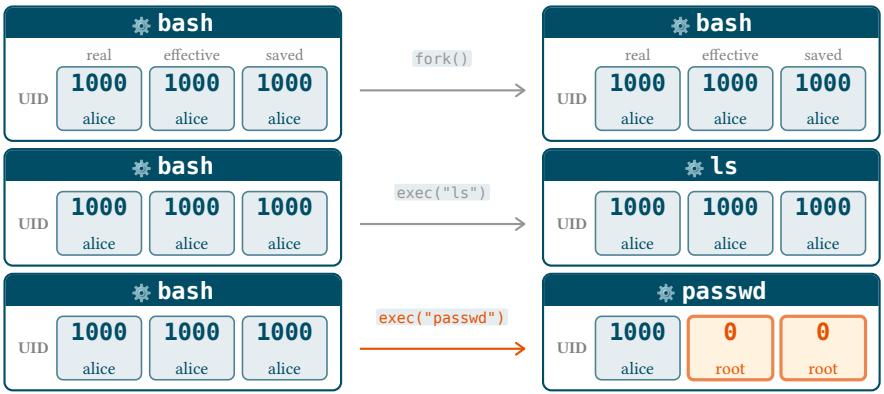
🧬 the child gets a **copy**

➖ a regular program: **nothing** changes

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Label Changes

🧬 a process gets its label from its parent



🧬 the child gets a **copy**

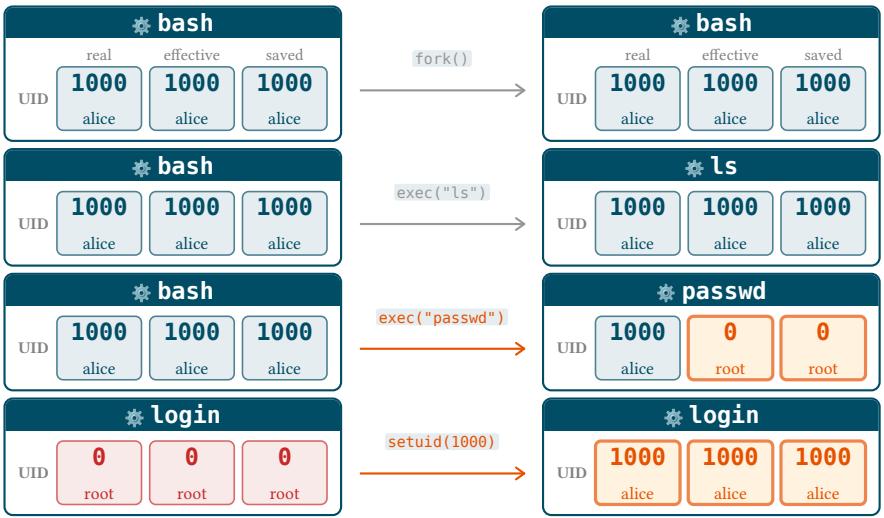
➖ a regular program: **nothing** changes

🪄 a `setuid` file: *effective* and *saved* become the **owner** of the file, `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Label Changes

🧬 a process gets its label from its parent



🧬 the child gets a **copy**

➖ a regular program: **nothing** changes

🪄 a `setuid` file: *effective* and *saved* become the **owner** of the file, `root`

👑 only `root` may: **all** the UIDs change, this is how `login`, `su` and `sudo` work

⚠️ after `setuid(1000)` no UID is `0` any more: the process can **never** go back to `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

The Prompt

who am I?

Bibliography

for this section

1. **Razvan Deaconescu, Razvan Rughinis, Mihai Carabas, Alexandru Radovici**, *Utilizarea Sistemelor de Operare*, Printech 2021, download
   - Chapter 5 - *Utilizatori*
   - Chapter 7 - *Interfața în linia de comandă*

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Prompt Shows the User

`bash` builds the prompt from the variable `PS1`

```
PS1='\u@\h:\w\$ '
```

```
alice@laptop:~/Movies$
```

Escape	Meaning	`alice`	`root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Prompt Shows the User

`bash` builds the prompt from the variable `PS1`

```
PS1='\u@\h:\w\$ '
```

```
alice@laptop:~/Movies$
```

Escape	Meaning	`alice`	`root`
`\u`	the **user**	`alice`	`root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Prompt Shows the User

`bash` builds the prompt from the variable `PS1`

```
PS1='\u@\h:\w\$ '
```

```
alice@laptop:~/Movies$
```

Escape	Meaning	`alice`	`root`
`\u`	the **user**	`alice`	`root`
`\h`	the **host** name	`laptop`	`laptop`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Prompt Shows the User

`bash` builds the prompt from the variable `PS1`

```
PS1='\u@\h:\w\$ '
```

```
alice@laptop:~/Movies$
```

Escape	Meaning	`alice`	`root`
`\u`	the **user**	`alice`	`root`
`\h`	the **host** name	`laptop`	`laptop`
`\w`	the current directory	`~/Movies`	`/etc`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Prompt Shows the User

`bash` builds the prompt from the variable `PS1`

```
PS1='\u@\h:\w\$ '
```

```
alice@laptop:~/Movies$
```

Escape	Meaning	`alice`	`root`
`\u`	the **user**	`alice`	`root`
`\h`	the **host** name	`laptop`	`laptop`
`\w`	the current directory	`~/Movies`	`/etc`
`\$`	`$` for users, `#` for UID `0`	`$`	`#`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

How the Prompt Shows the User

`bash` builds the prompt from the variable `PS1`

```
PS1='\u@\h:\w\$ '
```

```
alice@laptop:~/Movies$
```

```
root@laptop:/etc#
```

Escape	Meaning	`alice`	`root`
`\u`	the **user**	`alice`	`root`
`\h`	the **host** name	`laptop`	`laptop`
`\w`	the current directory	`~/Movies`	`/etc`
`\$`	`$` for users, `#` for UID `0`	`$`	`#`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Do Not Trust the Prompt



🎭 the prompt is just **text**, any user can change it

```
alice@laptop:~$ PS1='root@laptop:~# '
```

- `alice` sets her prompt to look like `root`

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Do Not Trust the Prompt



🎭 the prompt is just **text**, any user can change it

```
alice@laptop:~$ PS1='root@laptop:~# '
root@laptop:~# whoami
alice
```

- `whoami` asks the **kernel**: the shell runs as `alice`

> ✅ **ask the kernel**
>
> `whoami` and `id` show the **real** label of the process

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Do Not Trust the Prompt

🎭 the prompt is just **text**, any user can change it

```
alice@laptop:~$ PS1='root@laptop:~# '
root@laptop:~# whoami
alice
root@laptop:~# id
uid=1000(alice) gid=1000(alice) groups=1000(alice)
```

- `id` shows the whole label: UID, GID and groups

> ✅ **ask the kernel**
>
> `whoami` and `id` show the **real** label of the process

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

Do Not Trust the Prompt



🎭 the prompt is just **text**, any user can change it

```
alice@laptop:~$ PS1='root@laptop:~# '
root@laptop:~# whoami
alice
root@laptop:~# id
uid=1000(alice) gid=1000(alice) groups=1000(alice)
root@laptop:~# cat /etc/shadow
cat: /etc/shadow: Permission denied
```

- the kernel still checks the **UID**: `alice` is not `root`

> ✅ **ask the kernel**
>
> `whoami` and `id` show the **real** label of the process

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0

We talked about

- Users
- The Users Database
- Local Password Storage
- Groups
- Logging In
- The `root` User
- Becoming `root`
- File Permissions
- Special Permissions
- The Label of a Process
- The Prompt

 Copyright (C) 2026 Wyliodrin & Politehnica Bucharest, CC-BY-SA 4.0