Users and Groups

Job Readiness Companion

Turn a vague statement like “permissions are wrong” into something concrete that can be checked. The goal is to identify which users and groups exist, what they can reach, and what actions they can actually perform.

Common artifacts you would see

- Security findings that reference a specific file or directory, and note that access is broader than intended
- Operations tickets asking to add someone to a group to unblock work
- Audit notes pointing out shared accounts, old group memberships, or unused users
- Change requests that propose tightening permissions and ask for proof that work still functions
- Incident reviews where the central question becomes which account could have accessed or changed something

Junior expectations

- Name the exact user and group involved instead of saying “a user” or “the team.”
- Identify the exact file or directory, including the full path
- Record who owns the file or directory and which group it belongs to
- Translate permissions into plain language, such as who can read, who can write, and who can enter
- Treat directories carefully and distinguish between listing names, entering the directory, and modifying contents
- Test access by acting as the user instead of guessing based on output
- Separate what you observed from what you expected to happen
- Make small, targeted permission changes and retest after each one

Junior guardrails

- Do not treat octal numbers as the conclusion. Always translate them into actions and identities.
- Do not say “the group has access” without naming the group and the action
- Do not confuse directory execution with running a program. On directories, it controls whether you can enter
- Do not add users to groups as a shortcut without stating what it expands access to

- Do not widen permissions just to make something work without writing down the new trust decision
- Do not rely on a single command output. Verify by attempting the action

How issues are discussed professionally

In operations, audits, and security reviews, access issues are discussed as:

Identity
→ Object
→ Allowed actions
→ Proof by test
→ boundary decision.

A credible explanation sounds like: “User Maya (groups: support, oncall) can read `/srv/reports/quarterly.csv` because the file is owned by group support and is group-readable. The directory `/srv/reports` is also group-executable, so she can enter it and open the file. That access is broader than intended because quarterly financials were expected to be limited to the finance team. The observable impact is that non-finance users can retrieve the report, and downstream systems may ingest data outside the intended audience.”

Not: “The permissions are wrong.”

Your job is to make the first statement possible.

Translation patterns you must be able to use

Unintended read access

Structure:

- Identity (user and relevant groups)
- Object (file/directory path)
- Control (owner/group/mode/ACL)
- Allowed action (what read means in practice)
- Proof (how you verified as that identity)
- Observable damage (what would be noticed if accessed)

Example:

> “User `jen` (groups: eng, contractors) can read /etc/app/secrets.env because the file is group-readable by contractors and the parent directory is executable for that group. I verified by switching to `jen` and running a direct read. The observable impact is that database credentials can be copied into local scripts and used to access production data outside approved workflows.”

Unintended write access

Structure:

- Identity
- Object and location (path and parent directory)
- Write mechanism (file writable, directory writable, or ACL grants)
- Consequence path (what that write can influence: config, executable, logs, state)
- Proof (test the actual write action)
- Observable damage (what changes, breaks, or becomes untrustworthy)

Example:

> “User `alex` (groups: ops) can modify /usr/local/bin/backup because the file is group-writable by ops. That write access influences what runs in the nightly backup job, which executes the script as root. I verified by attempting a harmless edit under `alex` and confirming the checksum changed. The observable impact is that privileged behaviour can be altered without a change record, visible as unexpected file modifications and backup destinations changing.”

Directory confusion (enter vs list vs modify)

Structure:

- Identity
- Directory path
- Permission meanings (read=list names, execute=enter, write=create/delete)
- What the identity can actually do in practice
- Proof (actions attempted)
- Observable damage (what leaks or what can be altered)

Example:

> “User `sam` can enter /srv/cases but cannot list it because the directory is `--x` for his group. That means he can access known filenames if he guesses them or receives a path from elsewhere, even though ls appears empty. I verified by attempting cd and opening a known case file by name. The observable impact is that ‘can’t list’ may be mistaken for ‘can’t access,’ leading to unintended disclosure of specific records.”

How this skill is used on the job

Operations support and access provisioning

- Question this helps answer: “What access is needed for work to function, and what does it expand?”
- Inputs you work from: ticket requests, group membership, file paths, service accounts
- What your explanation enables: minimal changes (least privilege) that unblock work without broadening trust unintentionally

Security findings and remediation

- Question this helps answer: “Who can do what to which object, and why is that a risk?”
- Inputs you work from: permission snapshots, ACLs, ownership, group inventories
- What your explanation enables: targeted tightening tied to specific identities and observable consequences

Incident reviews and accountability

- Question this helps answer: “Which identities could have accessed or changed this, and how do we know?”
- Inputs you work from: file ownership/modes, access logs, user/group records, timelines
- What your explanation enables: narrowing plausible actors and strengthening controls to prevent recurrence

Interview translation

Question:

> “How do you assess and explain a permissions or access control issue on Linux?”

Answer:

> “I start by naming exactly what I am looking at. I identify the file or directory and check who owns it and which group it belongs to. Then I look at the user trying to access it and check which groups they belong to. After that, I translate the permissions into plain language so I can say who can read it, who can write to it, and who can enter the directory if it is one.
>
> I do not assume my interpretation is correct. I test it by switching to that user and trying the action directly. If the result does not match my expectations, I update my understanding based on what actually happened. Once I can clearly explain who can do

what, I decide whether that access makes sense. If it does not, I suggest a slight change and test again to ensure regular operation still works."

Answer pattern:

- Name the object (file/directory) and its full path
- Identify the owner and group
- Identify the user and their groups
- Translate permissions into actions (read/write/enter/create/delete)
- Test by performing the action as that user
- Explain the mismatch (if any) and update understanding
- Suggest a small, targeted change and retest

Common early-career mistakes

- Treating octal output as the explanation instead of translating it into identities and actions
- Checking permissions but not testing the action as the actual user (or equivalent)
- Confusing directory permissions and concluding "no access" because the listing fails
- Unblocking work by widening permissions or adding to groups without recording the trust decision
- Fixing the symptom (chmod/chgrp) without stating which concrete action is intended to change

What good looks like / If you forget everything else

- You can name the identity (user and groups) and the object (full path)
- You can translate permissions into real actions: read, write, enter, create, delete
- You verify by attempting the action as that identity, not by trusting a single command
- You keep the scope tight and make the trust decision explicit
- Your change is minimal, reversible, and validated by retesting

If a teammate can repeat your explanation and reach the same conclusion, the work is job-ready.