# Advanced Programming in the UNIX Environment

## Week 13, Segment 2:
eUIDs, file flags, mount options, securelevels

**Department of Computer Science**
**Stevens Institute of Technology**

**Jan Schaumann**
jschauma@stevens.edu
https://stevens.netmeister.org/631/

## Changing eUIDs

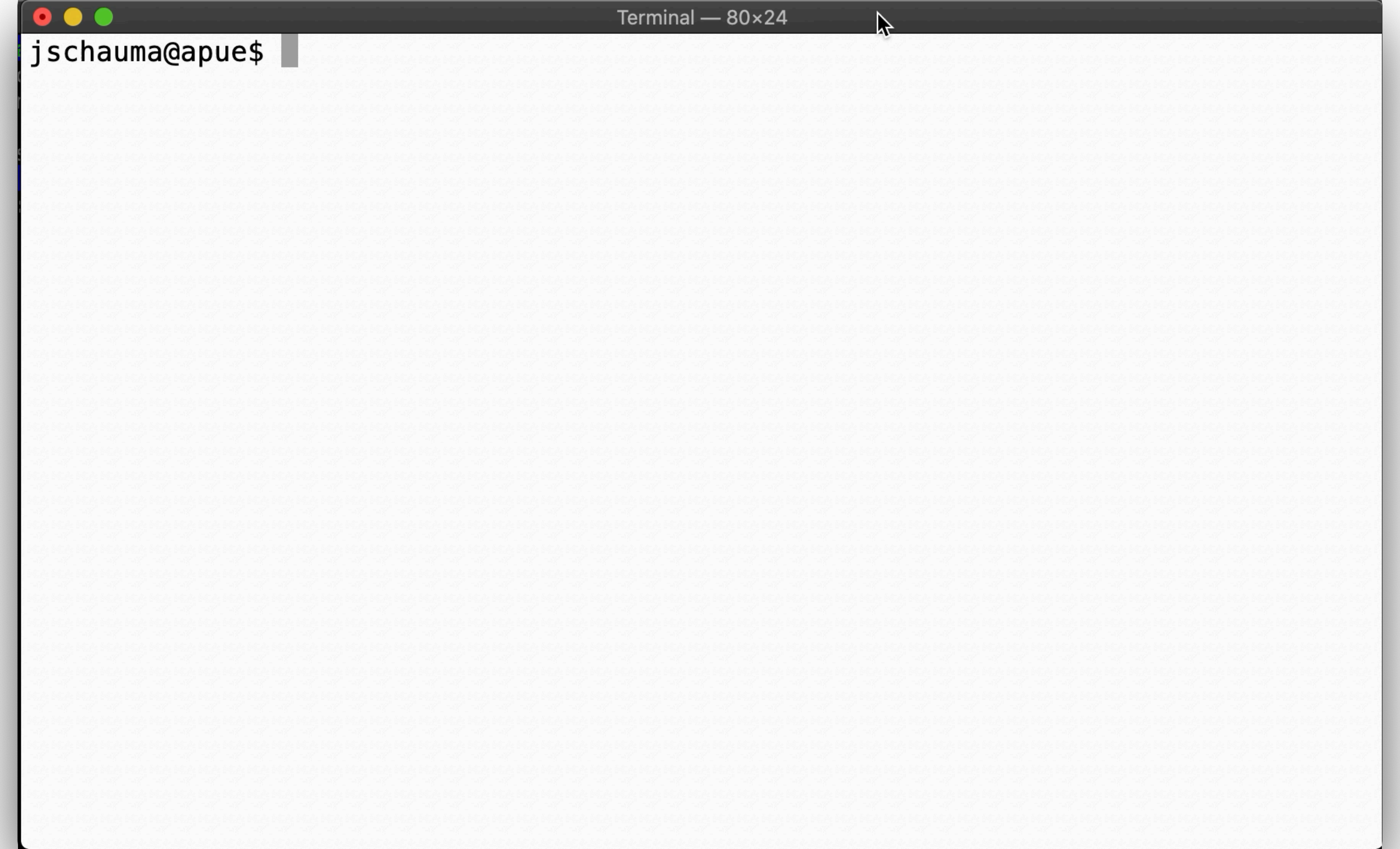
ACLs control access to files and directories by eUID/eGID. Recall from Week 03, Segment 2 that we can change those: `setuid.c`

Common examples:

- necessary access to privileged resources (*e.g.*, binding to a port<1024, use of raw sockets for ICMP, …)
- handling logins (*e.g.*, `login(1)`, `sshd(8)`)
- *raising* and *changing* privileges (*e.g.*, `su(1)`, `sudo(8)`)

## Pitfalls when changing eUIDs

- setuid programs
  - require careful raising and lowering privileges *only when needed* (Least Privilege)
  - rely on correct ownership and permissions (*i.e.*, factors outside of the control of the program)
- `su(1)`
  - requires sharing of a password
  - grants all or nothing access
- `sudo(8)`
  - often misconfigured granting too broad access (`ALL:ALL`)
  - additional authentication often dropped (`NOPASSWD`)
  - restrictions often overlook privilege escalation



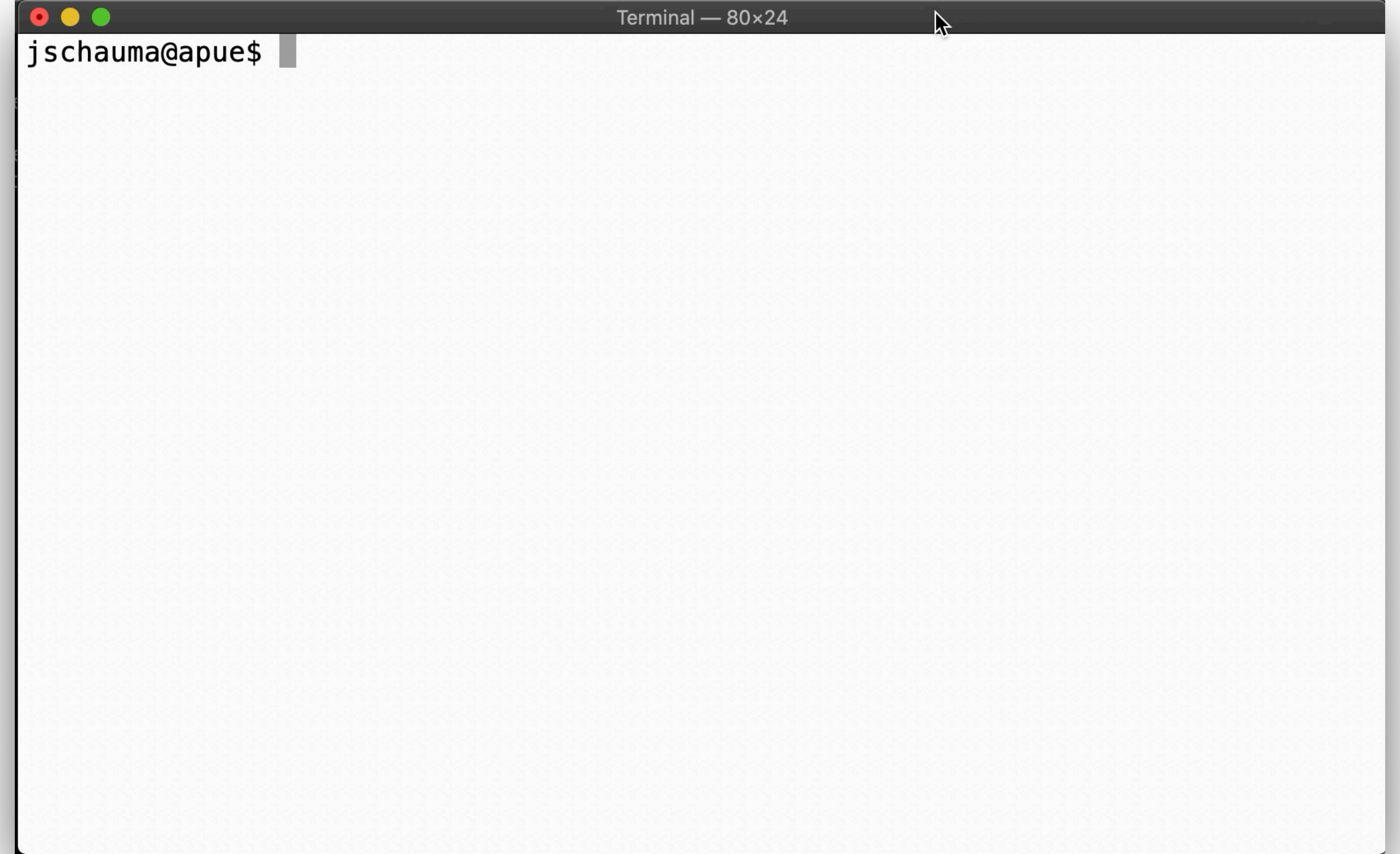
```
jschauma@apue$
```

## chflags(2)

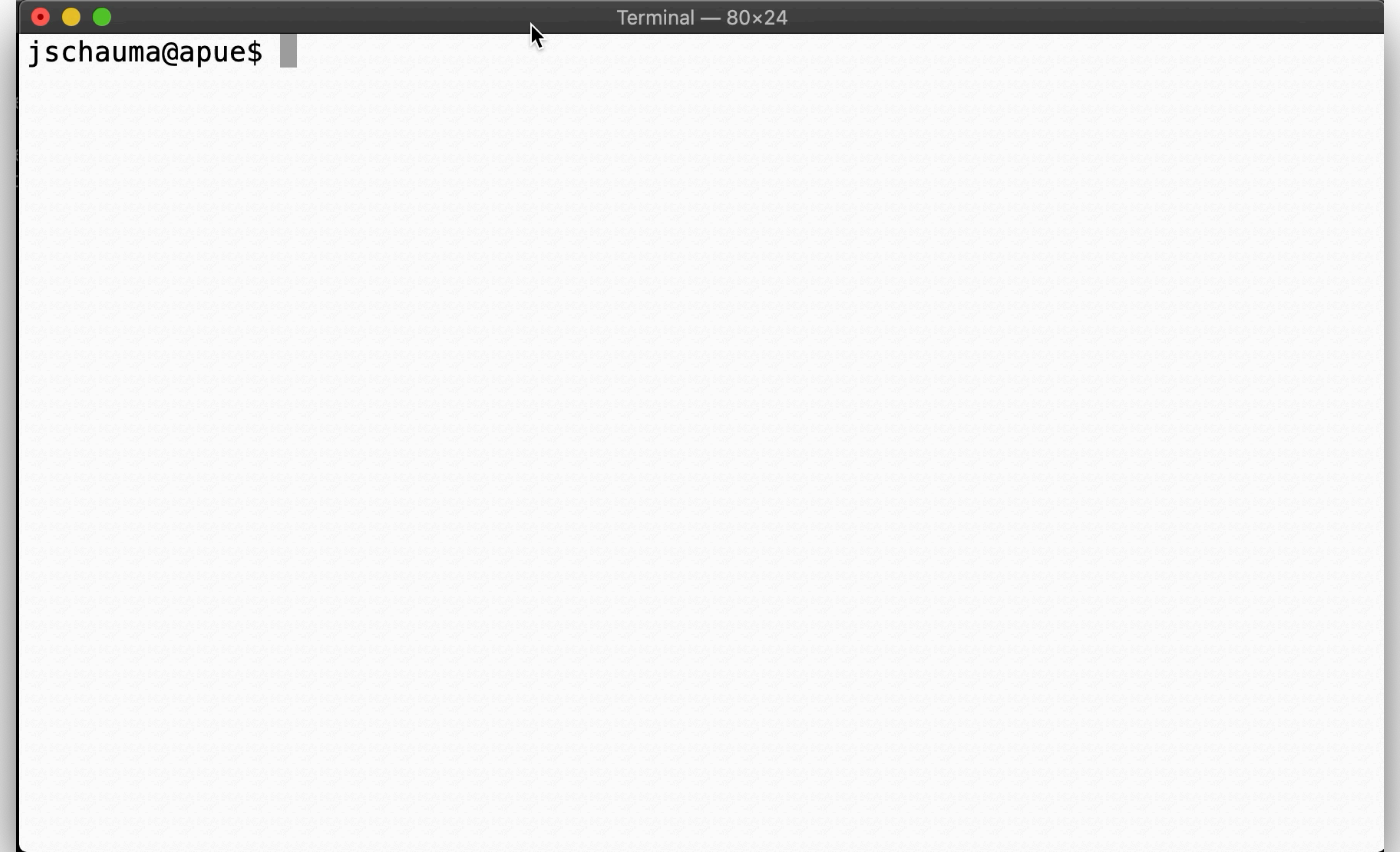
```
#include <sys/stat.h>
#include <unistd.h>
int chflags(const char *path, u_long flags);
int lchflags(const char *path, u_long flags);
int fchflags(int fd, u_long flags);
                                        Returns: 0 on success, -1 on error
```

Your eUID controls access to resources. But we can restrict certain access further via *e.g.*, “file flags”:

| | |
|---|---|
| `UF_APPEND` | The file may only be appended to. (owner or super-user) |
| `UF_IMMUTABLE` | The file may not be changed. (owner or super-user) |
| `SF_APPEND` | The file may only be appended to. (super-user only) |
| `SF_IMMUTABLE` | The file may not be changed. (super-user only) |



```
jschauma@apue$
```

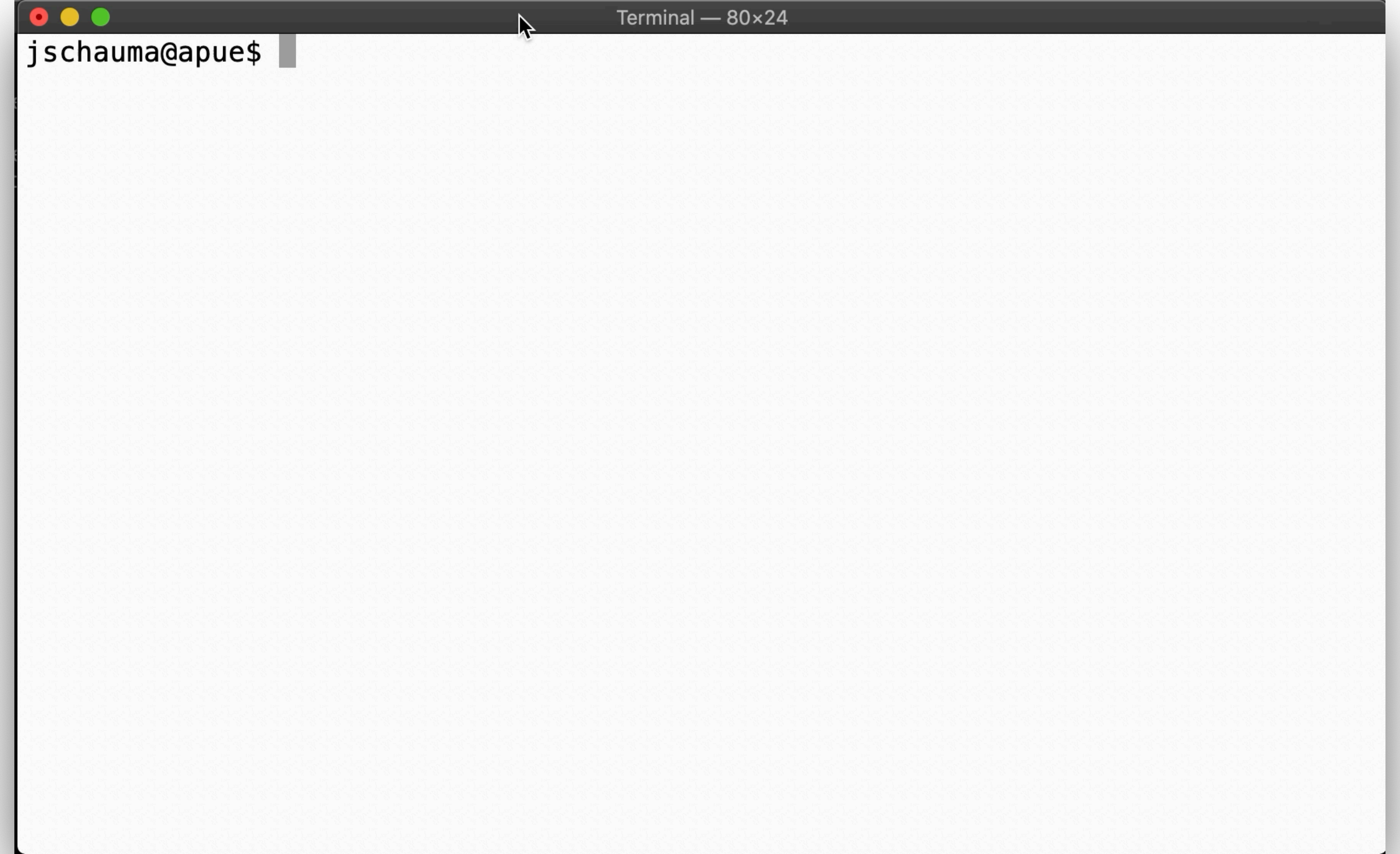


```
jschauma@apue$ 
```

## securelevels

To prevent even eUID 0 from *e.g.*, changing the mount flags, you can employ *securelevels*:

- superuser can raise the securelevel, only `init(8)` can lower it
- in other words, lowering requires a reboot
- four securelevels are defined
  - `-1` "Permanently insecure mode"
  - `0` "Insecure mode"
  - `1` "Secure mode"
  - `2` "Highly secure mode"
- see `secmodel_securelevel(9)`



```
jschauma@apue$
```

## Summary

- `su(1)` and `sudo(8)` can be used to grant others the ability to run commands as another user, but it can be difficult to restrict access
- “file flags” may restrict certain use; see `chflags(1)`/`chflags(2)` on BSD, `chattr(1)` on Linux
- mount options like `noexec`, `nosuid`, `rdonly` can restrict and protect filesystems per mount point
- to prevent even root from undoing these protections, use `securelevels` (reboots are noisy)