Advanced Programming in the UNIX Environment

Week 04, Segment 3: Directories

Department of Computer Science
Stevens Institute of Technology

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

mkdir(2)

```
#include <sys/stat.h>
#include <fcntl.h>

int mkdir(const char *path, mode_t mode);
int mkdirat(int fd, const char *path, mode_t mode);
```

Returns: 0 on success, -1 on error

- creates a new, empty (except for . and .. entries) directory
- access permissions specified by mode and restricted by the umask(2) of the calling process
- ownership as previously discussed (st_uid = euid; st_gid = st_gid of directory it is created in, or st_gid = egid)

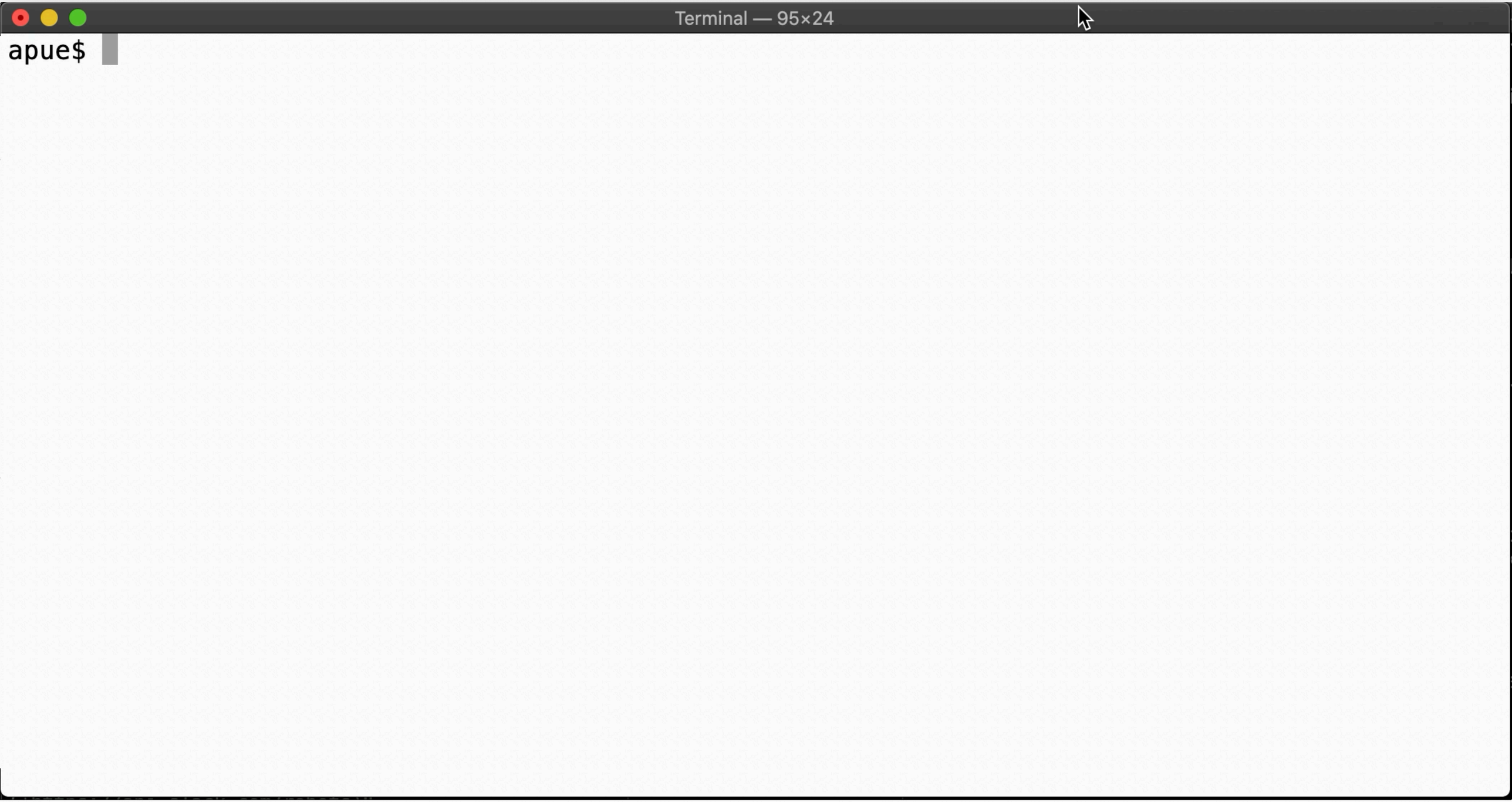
rmdir(2)

```
#include <unistd.h>

int rmdir(const char *path);
```

Returns: 0 on success, -1 on error

- removes the given directory if:
  - the directory is empty (except for . and ..)
  - st_nlink is 0 (after this call)
  - no other process has the directory open


Terminal — 95×24
apue$

Terminal — 80×24

```
main(int argc, char **argv) {

        DIR *dp;
        struct dirent *dirp;

        if (argc != 2) {
                fprintf(stderr, "usage: %s dir_name\n", argv[0]);
                exit(EXIT_FAILURE);
        }

        if ((dp = opendir(argv[1])) == NULL) {
                fprintf(stderr, "Unable to open '%s': %s\n",
                                        argv[1], strerror(errno));
                exit(EXIT_FAILURE);
        }

        while ((dirp = readdir(dp)) != NULL) {
                printf("%s\n", dirp->d_name);
        }

        (void)closedir(dp);
        return EXIT_SUCCESS;
}
apue$
```

Reading directories

```
#include <dirent.h>

DIR *opendir(const char *path);
DIR *fdopendir(int fd);
```

Returns: pointer if OK, NULL on error

```
struct dirent *readdir(DIR *dirp);
```

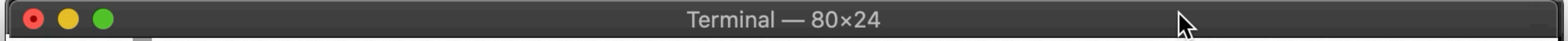
Returns: pointer to next entry if OK, NULL on end of directory or error

```
int closedir(DIR *dirp);
```

Returns: 0 on success, -1 on error

Reading directories

- `opendir(2)` / `readdir(2)` requires *read* permissions, while opening a file inside a directory requires *exec* permissions on the directory
- the format of directory entries is filesystem and implementation dependent; use `readdir(2)` / `getdents(2)` -- see `dirent(3)`
- the type `DIR` represents a directory *stream*; an ordered sequence of all directory entries in a particular directory
- file descriptor limitations *may* (or may *not*) apply to directory stream; see `COMPATIBILITY` in `opendir(2)`
- for directory traversal, prefer *e.g.*, `fts(3)`, if available




```
apue$
```

Moving around directories

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

Returns: buf if OK, NULL on error

Get the kernel's idea of our process's current working directory.

```
$ pwd
/home/jschauma
$ cd /tmp
$ echo $PWD
/tmp
```

Moving around directories

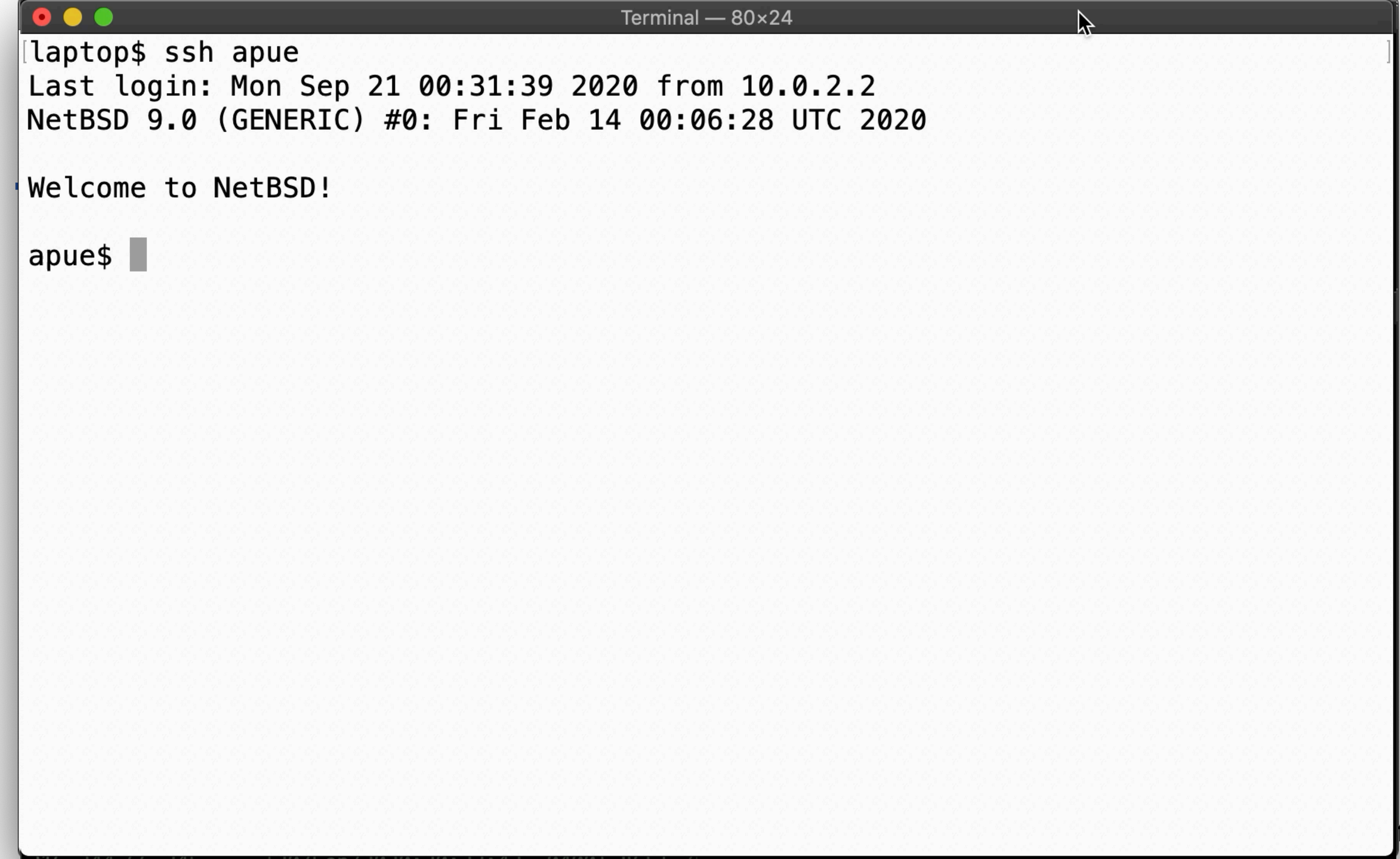
```
#include <unistd.h>

int chdir(const char *path);
int fchdir(int fd);
```

Returns: 0 if OK, -1 on error

Change the process's current working directory.

Requires exec permissions on the directory in question.


Terminal — 80×24
laptop$ ssh apue
Last login: Mon Sep 21 00:31:39 2020 from 10.0.2.2
NetBSD 9.0 (GENERIC) #0: Fri Feb 14 00:06:28 UTC 2020
Welcome to NetBSD!
apue$

Directories

No surprises here:

- `mkdir(1)` uses `mkdir(2)`
- `rmdir(1)` uses `rmdir(2)`

`opendir(2)` / `readdir(2)` are nice, but you want `fts(3)` for proper file hierarchy traversal.

`cd(1)` must be a shell builtin.

Coming up: what's the size of a directory, anyway?