Each user fits into only one of the three permission sets. The permissions used are those that are most specific. For example, the owner of a file always has access determined by the owner permission bits and never the group permission bits. It is possible for the "other" and "group" categories to have more access than the owner, although this configuration would be highly unusual.

On a regular file, the read bit allows the file to be opened and read. The write bit allows the contents of the file to be modified or truncated; however, the ability to delete or rename (or delete and then recreate!) the file is controlled by the permissions on its parent directory because that is where the name-to-dataspace mapping is actually stored.

The execute bit allows the file to be executed. Two types of executable files exist: binaries, which the CPU runs directly, and scripts, which must be interpreted by a shell or some other program. By convention, scripts begin with a line similar to

```
#!/usr/bin/perl
```

that specifies an appropriate interpreter. Nonbinary executable files that do not specify an interpreter are assumed to be **bash** or **sh** scripts.[8]

For a directory, the execute bit (often called the "search" or "scan" bit in this context) allows the directory to be entered or passed through while a pathname is evaluated, but not to have its contents listed. The combination of read and execute bits allows the contents of the directory to be listed. The combination of write and execute bits allows files to be created, deleted, and renamed within the directory.

A variety of extensions such as access control lists (see page 159), SELinux (see page 923), and "bonus" permission bits defined by individual filesystems (see page 158) complicate or override the traditional nine-bit permission model. If you're having trouble explaining the system's observed behavior, check to see whether one of these factors might be interfering.

The setuid and setgid bits

The bits with octal values 4000 and 2000 are the setuid and setgid bits. When set on executable files, these bits allow programs to access files and processes that would otherwise be off-limits to the user that runs them. The setuid/setgid mechanism for executables is described on page 106.

When set on a directory, the setgid bit causes newly created files within the directory to take on the group ownership of the directory rather than the default group of the user that created the file. This convention makes it easier to share a directory of files among several users, as long as they belong to a common group. This interpretation of the setgid bit is unrelated to its meaning when set on an executable file, but no ambiguity can exist as to which meaning is appropriate.

8. The kernel understands the #! ("shebang") syntax and acts on it directly. However, if the interpreter is not specified completely and correctly, the kernel will refuse to execute the file. The shell then makes a second attempt to execute the script by calling **sh**.