File Permissions

- An access control mechanism
- Based on relation between file & user
- Analogy:
  - Documents receive classification
  - Employees receive clearance
  - Access to a particular document is determined by the documents classification and the employees clearance

File Permissions

- A file has 3 modes of access:
  - Read (r) - Can view the file
  - Write (w) - Can change the file
  - Execute (x) - Can run the file (program)

File Permissions

- A file can be accessed by 3 different types of people:
  - The file owner or user (u)
  - A member of the files group (g)
  - Anyone else or others (o)

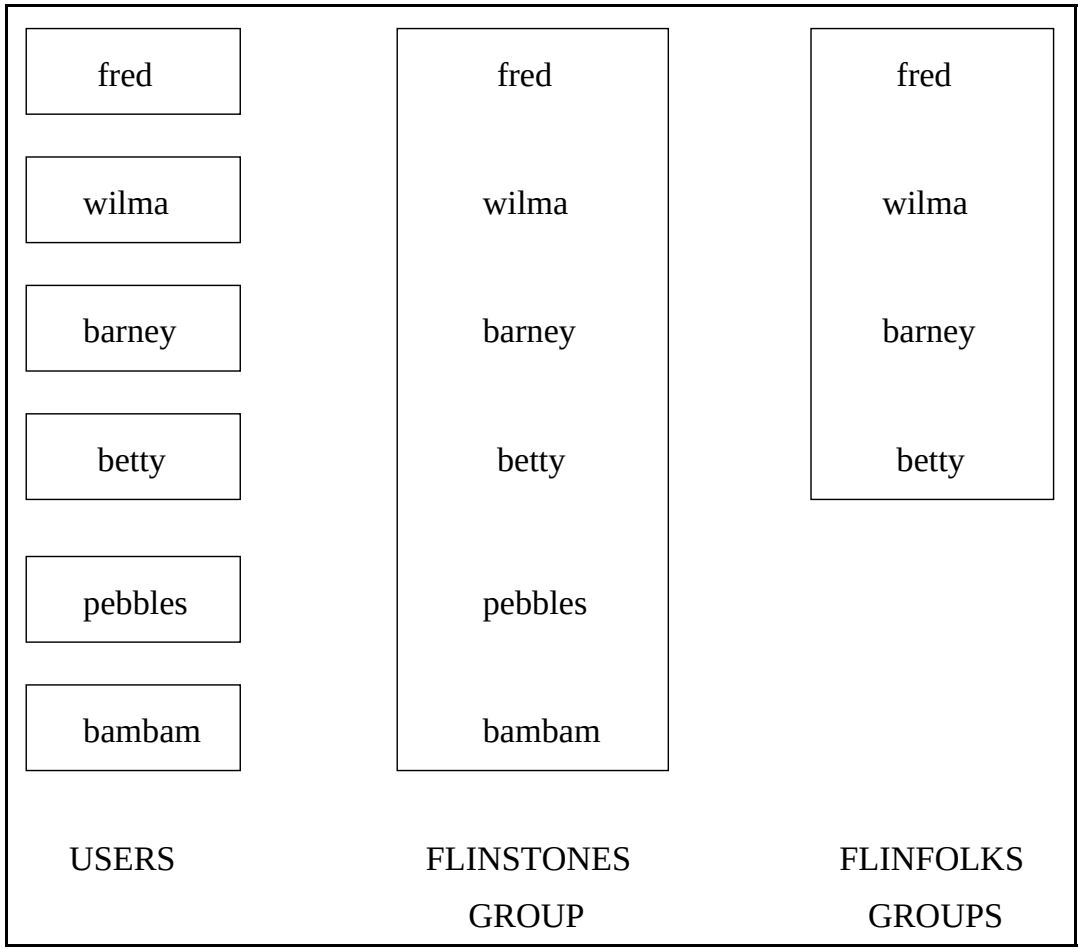
Directory Permissions

- Directories are treated in the same way as files
- They have an associated owner
- They have an associated group
- The permissions do slightly different things
  - Read (r) - Can view the contents of directory (ls)
  - Write (w) - Can add, delete, rename files
  - Execute (x) - Can ’cd’ into the directory and open files in it or its subdirectories

USERS & GROUPS

- A user is any one person (one & only one)
- A group consists of one or more users
- A user may be a member of more than one group

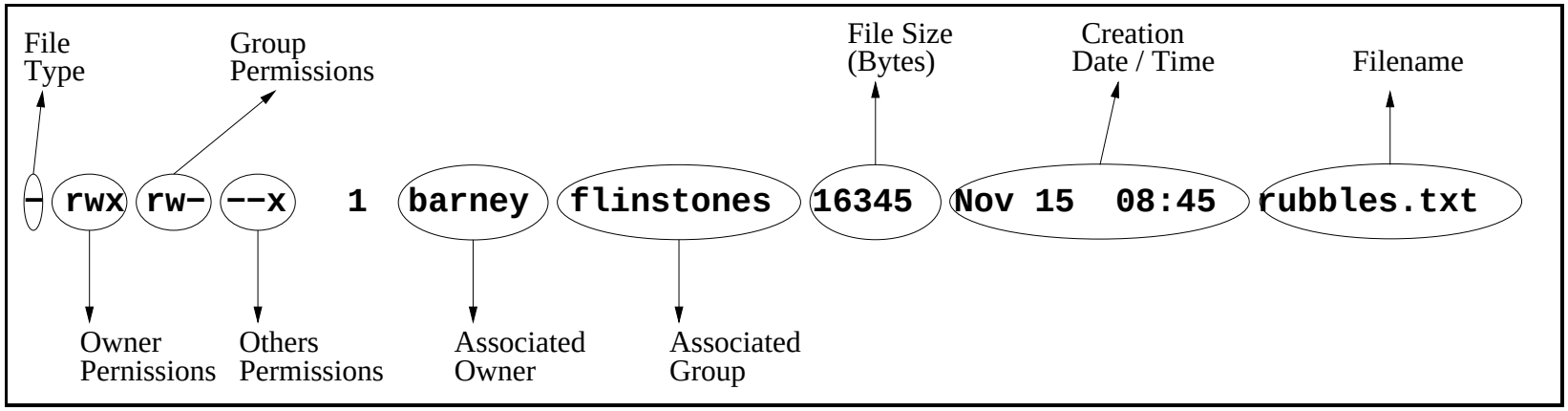
USERS & GROUPS



ls -l is your friend

- All of the file's attributes can be examined using the `ls -l` command

```
$ ls -l rubbles*
-rwxrw---x  1 barney  flinstones  16345  Nov15 08:45 rubbles.txt
$
```



Numeric Equivalents

- Each of the permission bits are bitmapped as follows:

Special Bits			User (U)			Group (G)			Other (O)		
SUID (S/s)	SGID (S/s)	Sticky (T/t)	Read (r)	Write (w)	Exec (x)	Read (r)	Write (w)	Exec (x)	Read (r)	Write (w)	Exec (x)
4000	2000	1000	400	200	100	40	20	10	4	2	1

r	w	x	Value
-	-	-	0
-	-	x	1
-	w	-	2
-	w	x	3
r	-	-	4
r	-	x	5
r	w	-	6
r	w	x	7

chown & chgrp

- A file's owner can be changed using chown:

```
# ls -l rubble.txt
-rw-rw-r--    1 barney   flinstones ...  rubble.txt

# chown fred rubble.txt
# ls -l rubble.txt
-rw-rw-r--    1 fred     flinstones ...  rubble.txt
```

chown & chgrp

- A file's owner & group can also be changed using chown:

```
# ls -l rubble.txt
-rw-rw-r--    1 barney   flinstones ...  rubble.txt

# chown fred:flinfolks rubble.txt
# ls -l rubble.txt
-rw-rw-r--    1 fred     flinfolks  ...  rubble.txt
```

chown & chgrp

- To change only the group use chgrp:

```
# ls -l rubble.txt
-rw-rw-r--    1 barney   flinstones ...  rubble.txt

# chgrp flinfolks rubble.txt
# ls -l rubble.txt
-rw-rw-r--    1 barney   flinfolks  ...  rubble.txt
```

chmod

- chmod is used to change file permissions
- Permissions can be specified:
  - In absolute form - Use octal specification
  - Surgicaly - Use who/how/what specification

chmod - Octal specification

When using an octal specification, you must set the permissions for each of the user, group and other in one go:

```
$ chmod 0543 test.txt
$ ls -l test.txt
-r-xr---wx    1 andy     andy  ...  test.txt
```

chmod - who/how/what specification

Who may be one of:

- u - The file's owner (user)
- g - The file's group
- o - Other users (world)
- a - All three of them

chmod - who/how/what specification

How may be one of:

- + Add permission, existing unaffected
- - Remove permission, existing unaffected
- = Set permission, existing replaced

chmod - who/how/what specification

What may be one of:

- r - Read permission
- w - Write permission
- x - Execute permission

chmod - what specification

Some examples:

Add execute permission for the file's owner (and leave everything else)

```
# chmod u+x file.txt ↩
```

Remove write permission from group and others (and leave everything else)

```
# chmod go-w file.txt ↩
```

Set the file to read only for everyone (kills existing permissions)

```
# chmod a=r file.txt ↩
```

umask

- When a file is created, the system needs to know what permissions to assign to the newly created file. This is done using 'umask'
- You set the bits in umask that you **dont** want set on any newly created file.
- A newly created file will **never** have the execute bit set, regardless of the value of umask.
- For example, a umask of 0022 will ensure that write access is not granted to group and others.

```
$ umask 0022
$ touch test.txt
$ ls -l test.txt
-rw-r--r--    1 andy     andy   ... test.txt
```

Setuid bit (4000)

The setuid bit is represented by a ’S’ in the user/executable field in the file permissions if the file is not executable or by a ’s’ in that field if the file is executable:

```
-rwSrw-rw-    --> Setuid bit set, not executable
-rwsrw-rw-    --> Setuid bit set, executable
```

Setuid bit (4000)

The setuid bit is only used for files:

Files:

The user executing the file gains the privileges of the file's owner for the duration of that process' run life. For example, a program owned by root with the setuid bit set `(setuid root)` when run by a normal user will gain root privileges for the purposes of that process. It changes the effective user. One exception: Setuid is ignored if the executable file is a script (security)

Directories:

The setuid bit is ignored completely on directories and does SFA

Setuid bit - Example

```
$ ls -l hexdump
-rwxr-xr-x    1 root     root ...  hexdump
$ ls -l /dev/hda1
brw-rw----    1 root     disk ...  /dev/hda1
$ hexdump -n 10 /dev/hda1
 hexdump: /dev/hda1: Permission denied

# chmod 4755 hexdump
# ls -l hexdump
-rwsr-xr-x    1 root     root ...  hexdump

$ hexdump -n 10 /dev/hda1
0000000 ace9 4100 4a50 5726 1a4e
```

Setgid bit (2000)

The setgid bit is represented by a ’S’ in the group/executable field in the file permissions if the file is not executable or by a ’s’ in that field if the file is executable:

```
-rw-rwSrw-    --> Setgid bit set, not executable
-rw-rwsrw-    --> Setgid bit set, executable
```

Setgid bit (2000)

The setgid bit takes on a different meaning for files & directories:

Files:

The user executing the file gains the privileges of the file's group for the duration of that process' run life. For example, a program with an associated gruop of root with the setgid bit set `(setgid root)` when run by a normal user will gain group root privileges for the purposes of that process. It changes the effective group. One exception: Setgid is ignored if the executable file is a script (security)

Directories:

Any newly created file under a directory with the setgid bit set will have the group set to that of the group owner of the directory rather than the users default group.

Setgid bit - Example

```
$ ls -l hexdump
-rwxr-xr-x    1 root     root ...  hexdump
$ ls -l /dev/hda1
brw-rw----    1 root     disk ...  /dev/hda1
$ hexdump -n 10 /dev/hda1
hexdump: /dev/hda1: Permission denied

# chmod 2755 hexdump
# ls -l hexdump
-rwxr-sr-x    1 root     root ... hexdump

$ hexdump -n 10 /dev/hda1
hexdump: /dev/hda1: Permission denied
```

Setgid bit - Example

```
# chgrp disk hexdump
# ls -l hexdump
-rwxr-sr-x    1 root     disk .... hexdump

$ hexdump -n 10 /dev/hda1
0000000 ace9 4100 4a50 5726 1a4e
```

Sticky bit (1000)

The sticky bit is represented by a ’T’ in the others/executable field in the file permissions if the file is not executable or by a ’t’ in that fied if the file is executable:

```
-rw-rw-rwT    --> Sticky bit set, not executable
-rw-rw-rwt    --> Sticky bit set, executable
```

Sticky bit (1000)

The sticky bit takes on a different meaning for files & directories:

Files:

Keep programs in swap even after execution. (Historical, not really useful but maintained for backward compatability)

Directories:

Files in a directory with the sticky bit set can not be deleted by anyone other than:

- The owner of the file
- The owner of the directory
- The root user

Sticky bit Example)

```
[andy@Node4] tmp]$ ls -ld /tmp
drwxrwxrwt   27 root      root    ... /tmp
[andy@Node4] tmp]$ ls -l andy-temp
-rw-rw-rw-    1 andy      andy    ...  andy-temp

[patsy@Node4] tmp]$ cat andy-temp
This is Andy's file
[patsy@Node4] tmp]$ rm andy-temp
rm: cannot unlink 'andy-temp': Operation not permit-
ted

[andy@Node4] tmp]$ rm andy-temp
[andy@Node4] tmp]$
```