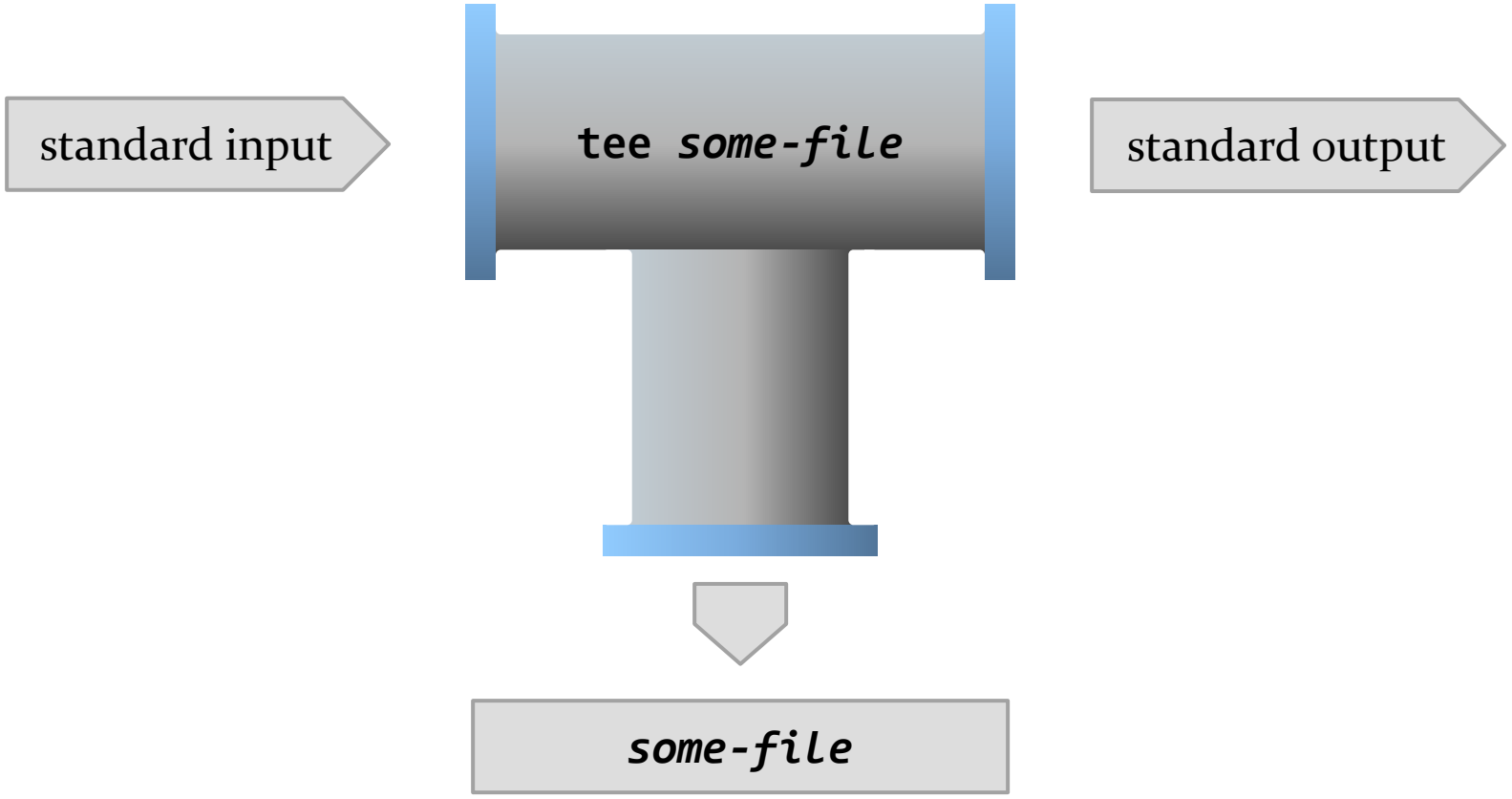
Tees; File permissions

Redirection and piping

- redirection allows you to send the output of a command to a file
- piping allows you to send the output of a command to the input of another command
- can you do both?
  - yes!

Tees

- the **`tee`** command reads standard input and copies it to both standard output and to one or more files



The tee command is useful for saving intermediate results in a pipeline to a file (or files)

- e.g., save the current date to a file and print the current year

```
date
date | tee fulldate.txt
date | tee fulldate.txt | cut --delimiter=" " --fields=1
date | tee fulldate.txt | cut --delimiter=" " --fields=2
date | tee fulldate.txt | cut --delimiter=" " --fields=6
date | tee fulldate.txt | cut -d" " -f6
```

You can insert a tee after any command in a pipline to record all of the intermediate outputs:

```
fortune | tee fortune.txt | cowsay | tee cow.txt \
 | cowsay - n
```

File permissions

- Unix-like OSes were designed for multi-tasking, multi-user systems
  - more than one user of the computer at the same time
    - users typically logged into the computer over a network
- for this to work, there need to be mechanisms in place that protect users from each other and that protect the OS from normal users
  - e.g., a user should not be able to modify (or perhaps even see) another user's files
  - e.g., a normal user should not be able to modify critical OS files

Owners

- a user can own files and directories
  - on your computer you own your home directory and all of the files inside your home directory
- the owner has control over access to their owned files
  - possible for the owner to grant access to a file to all other users (called the *world*) or to a group of users
    - however, only a system administrator can create new groups

The **`id`** command provides information about a user's identity

- the exact output will be different depending on your OS

```
id
id burton
```

Access rights

- a user has control of three different access rights for a file or directory
  1. read access
  2. write access
  3. execution access

Access rights

Access type	Files	Directories
read	File can be opened and read.	Contents can be listed if execute access on the directory is also granted.
write	File can be written to or truncated. Does not allow deletion or renaming.	Allows files in the directory to be created, deleted, or renamed if execute access on the directory is also granted.
execute	Allows the file to be run like a program or executed as a script.	Allows a directory to be entered.

The **`touch`** command changes the access and modification times of a file

- if the file does not exist then it creates an empty file

```
touch file1.txt
```

The **`mkdir`** command creates a directory

```
mkdir dir1
```

Use the long listing format option with **`ls`** to view the access rights (also called *permissions*) of files:

```
ls -l
ls -l dir1
ls -l file1.txt
```

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

file type

File types

Symbol	File type
`-`	Regular file
`d`	Directory
`l`	Symbolic link
`c`	Character special file (a device that handles data as a stream of bytes, e.g., a terminal)
`b`	Block special file (a device that handles data as blocks, e.g., a disk drive)

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

user read-write-execute permissions

Permissions

Symbol	File type
`r`	Read permission granted
`w`	Write permission granted
`x`	Execute permission granted
`-`	Permission not granted

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

group read-write-execute permissions

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

world read-write-execute permissions

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

number of hard links

Hard links

- a hard link is a directory entry that associates a filename with an actual file in the filesystem
  - must have at least one hard link for the original name for each file

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

owner name

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

group name

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

file size

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

modification/creation date and time

Long listing output

```
-rw-r--r-- 1 burton burton 1783 Sep  2 13:12 file1.txt
```

file/directory name

Changing permissions

- an owner of a file or directory can change the permissions using the **`chmod`** command
- permissions are specified using octal numbers or a symbolic representation

Changing permissions

Octal value	Binary value	File permissions
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Grant read access to user, remove all other permissions:

```
chmod 400 file1.txt
```

Grant read access to all users, remove all other permissions:

```
chmod 444 file1.txt
```

Grant all permissions to user, read and execute permissions for everyone else:

```
chmod 755 dir1
```

Symbolic representation

Symbol	Meaning
`u`	"User"/owner
`g`	"Group"
`o`	"Other"/world
`a`	"All" (`u`, `g`, and `o`)

Symbol	Meaning
`+`	Add a permission
`-`	Remove a permission
`=`	Assign a permission

Symbol	Meaning
`r`	Read
`w`	Write
`x`	Execute

Symbolic examples

Example	Meaning
`chmod +r fname`	Add read permission for all users to file **fname**
`chmod u+w fname`	Add write permission for owner to file **fname**
`chmod u-x fname`	Remove execute permission for owner to file **fname**
`chmod a=rwx fname`	Assign read, write, and execute permissions for all users to file **fname**
`chmod u+x,g=rx fname`	Add execute permission for owner, assign read and execute permissions for group for file **fname**

Expansions

- after typing a command and pressing Enter, Bash performs substitutions on the text before executing the command
  - for example, suppose that the current working directory contains the files **`a.txt`**, **`b.pdf`**, and **`c.out`**, then in the command

    `ls *`

    the **`*`** gets substituted by **`a.txt b.pdf c.out`** so the executed command is

    `ls a.txt b.pdf c.out`

Expansions

- the process of performing the substitutions is called expansion
- expansions involving wildcards is called pathname expansion

The command **`echo`** prints its string arguments to standard output

```
echo this is an echo
```

Pathname expansion occurs before the command is executed

```
touch file2.txt file3.txt fruit.exe
echo f*
echo file?.txt
echo fr*
echo G*
```