Permissions and Access Control

- Common Mistakes with Pipes
- Access Permissions
- Viewing Access Permissions
- The **chmod** Command
- Using **chmod** with Numeric Arguments

Common Mistakes with Pipes

- There are some mistakes that student seem to make each semester
- Two of these mistakes involve **pipes**
- The **first** mistake is to use ***cat*** at the beginning of a **pipeline** when it is **not** needed
  - Students will sometimes use ***cat*** to create input that is then piped into ***grep***, ***sort***, ***head*** or ***tail***
  - You don't need ***cat*** in a pipe with these commands because these utilities can read a file directly without using ***cat***

Common Mistakes with Pipes

- So instead of using

```
cat names.txt | grep Glenn
cat names.txt | sort
cat names.txt | head
cat names.txt | tail
```

- you should simply use

```
grep Glenn names.txt
sort names.txt
head names.txt
tail names.txt
```

- This mistake is harmless because it does not affect the result

Common Mistakes with Pipes

- However, the second mistake will cause an error
- The mistake is to use the filename given to the *first* command in a pipeline to the *following* commands
- If I wanted to find all Red Sox games where the Sox won at home I would write

```
grep Win red_sox.txt | grep vs
```

- I would **not** write

```
grep Win red_sox.txt | grep vs red_sox.txt  # WRONG
```

Access Permissions

- All Unix files and directories have **access permissions**
- The access permissions allow the **owner** of a file or directory to decide *who* gets to do *what* with the file or directory
- By default, the ***owner*** of a file or directory is the account that created it
- Every file, directory or device on a Unix filesystem has three types of permissions
  - ***Read***
  - ***Write***
  - ***Execute***

Access Permissions

- If you have **read permission** on a file, then you can look at the data in the file
- You can run ***cat***, ***more***, or ***less*** on these files
- If you *only* have read permission on a file, then you cannot change it
- To *change* a file, you need **write permission**
- To *run* a program or script file, you must have **execute permission**
- Each of the three types of permissions are set **either on or off** to three classes of users
  - The *owner*
  - The *group*
  - Every *other* Unix account

Access Permissions

- Every file or directory has an *owner* and a *group* assigned to it
- The account that created the file is usually the owner
- A **`group`** is a collection of Unix accounts
  - Every account also has a default group that is assigned to every file or directory that account creates
  - This default group is created when the account is created
  - Only a system administrator can add users to a group
  - Every file or directory is assigned to a group, though the owner can change this to another group
- The last class of users is any account that is not the owner or a member of the group. Unix calls this class of users "*other*"

Viewing Access Permissions

- To view the permissions of a file or directory use ***ls -l***

```
$ ls -l
total 5
-rw------- 1 it244gh libuuid 316 2011-09-20 21:32
   dead.letter
lrwxrwxrwx 1 it244gh libuuid  34 2011-09-06 13:21 it244 ->
   /courses/it244/s12/ghoffmn/it244gh
drwx------ 2 it244gh libuuid 512 2011-09-07 15:03 mail
drwxr-xr-x 2 it244gh libuuid 512 2011-09-25 15:48 test
-rw-r--r-- 1 it244gh libuuid  15 2011-09-20 16:18 test.txt
```

Viewing Access Permissions

- The *first* character indicates the type of file
  - A dash, **-** , means an ordinary file
  - The letter **d** indicates a directory
  - The letter **l** (el) indicates a link which we will discuss in the next class
- The *next three* characters indicate the access permissions of the owner
  - **r** means the owner has read permission
  - **w** means the owner has write (change) permission
  - **x** means the owner has execute (run) permission
  - **-** means the owner does not have the permission that would normally appear in this column

Viewing Access Permissions

- The *following* three characters (after the owner permissions) indicate the permissions of the group
- The *last* three characters are the permission of all other accounts
- The remaining columns provide file information:
  - After the permissions is a number that indicates the ***number of links*** to the file or directory
  - The following column is the ***owner*** of the file or directory
  - Next, you will find the ***group*** assigned to the file or directory
  - Following this is the ***size*** of the file in bytes
  - Next is the ***date and time*** the file or directory was created or last modified
  - The last column is the ***name*** of the file or directory

chmod

- When a file is created, is has certain default permissions

```
$ touch test.txt
$ ls -l test.txt
-rw-r--r-- 1 it244gh libuuid 0 2012-09-17 14:40 test.txt
```

- To change these permissions you use the ***chmod*** (**ch**ange **m**ode) command
- Only the ***owner*** of a file can do this

chmod

- ***chmod*** requires two arguments
  - The permissions you want to grant
  - The name of the file(s) or directory(s) which will be changed
- The format for a call to ***chmod*** is

```
chmod  PERMISSIONS  FILES_OR_DIRECTORIES
```

- The permission can be specified in two ways
  - Symbolically
  - Numerically

chmod

- *Symbolic* form uses letters and the plus and minus signs
- The *numeric* form uses three digits running from 0 to 7
  - I will teach the ***numeric*** format
  - The numeric format is initially confusing
  - But in my experience, it is the better way to go
- You are free to read about symbolic format in the textbook
- I will not deduct points for using this format in a test, quiz or homework assignment...as long as you use it ***correctly!***

Using *chmod* with Numeric Arguments

- The numeric permissions format uses three digits
- Each digit is a number from 0 to 7
  - *First digit*: gives the permissions of the ***owner***
  - *Second digit*: gives the permissions assigned to the ***group***
  - *Third digit*: gives the permissions for every ***other*** account
- Each of these classes of users must be assigned values for read, write, and execute permissions

Using *chmod* with Numeric Arguments

- How do you get three pieces of information out of one number?
  - By adding *powers of two*
  - Each digit is the ***sum*** of three other numbers
- When calculating the number, you add...
  - **4** if you want to give *read* permission
  - **2** if you want to give *write* permission
  - **1** if you want to give *execute* permission

Using *chmod* with Numeric Arguments

- Notice that all the number are powers of two
- If we write these values in binary notation
  - **100** represents **4**
  - **010** represents **2**
  - **001** represents **1**
- A single value from 0 to 7 is represented by 3 binary digits
- This is how we get *three* pieces of information from *one* digit

Using *chmod* with Numeric Arguments

- For example, to give full permissions I would add
  - **4** for read permission
  - **2** for write permission
  - **1** for execute permission
- So the total of 7 -- which is 111 in binary -- grants *all three permissions*
- Let's look at some other digits...

Using *chmod* with Numeric Arguments

- 6 in binary is 110
  - The leftmost digit is 1 indicating *read* permission
  - The center digit is 1 indicating *write* permission
  - The last digit is 0 indicating that *execute* permission is **not** granted
- 5 in binary is 101
  - The first digit is 1 so *read* permission is granted
  - The second digit is 0 so *write* permission is **not** granted
  - The last digit is 1 so *execute* permission is granted
- This scheme is confusing when you first encounter it
- But it becomes easier as you use it

Using *chmod* with Numeric Arguments

- Remember that you need *three* of these digits to specify the full permissions for a file or directory. Let's look at some examples...
- When you create a new file, it will have certain default permissions

```
$ touch foo.txt

$ ls
foo.txt

$ ls -l
total 0
-rw-r--r-- 1 it244gh libuuid 0 2012-02-09 15:51 foo.txt
```

- The *owner* can read and write the file, but not execute it
- The *group* and *everyone else* can only read the file

Using *chmod* with Numeric Arguments

- To make the file unreadable to everyone except the owner...

```
$ ls -l
total 0
-rw-r--r-- 1 it244gh libuuid 0 2012-02-09 15:51 foo.txt

$ chmod 600 foo.txt

$ ls -l
total 0
-rw------- 1 it244gh libuuid 0 2012-02-09 15:51 foo.txt
```

Using *chmod* with Numeric Arguments

- To change the file *back* to its default permissions...

```
$ ls -l
total 0

-rw------- 1 it244gh libuuid 0 2012-02-09 15:51
foo.txt
$ chmod 644 foo.txt

$ ls -l
total 0
-rw-r--r-- 1 it244gh libuuid 0 2012-02-09 15:51
foo.txt
```