CSE 390, Autumn 2012
Assignment 4: Users, Groups, Permissions
Due Tuesday, October 23, 2012, 12:30 PM

This assignment continues to practice using the `bash` shell and basics of combining commands using redirection and pipes. Electronically turn in **three** files: `homework4.txt`, `.plan`, and `.bash_profile` as described in this document. To receive credit, your `.plan` and `.bash_profile` should also be present in your home directory with proper permissions.

Task 1: .plan file (on attu)

A file named `.plan` can be placed into your home directory to tell other users of the system about you. In this file you can write any text you like, and the text will be displayed to other users when they use the `finger` command on you. Set up a `.plan` file for yourself on `attu` that contains the following contents, **you also need to turn this file in**.

- At least one sentence about yourself (can be real or made up)
- A cow message as would be output from the `~stepp/cowsay` or `~stepp/cowthink` programs
- One other piece of ASCII art, or a short message in large text
- Want more cows in your plan? Check out: http://www.sunnyspot.org/asciiart/gallery/cow.html

Test your `.plan` file by running `finger` on yourself. Turn in this file as part of your assignment submission, but part of this task is also making sure that the file can be seen by other students successfully when they use the `finger` command. The file must have read permissions for all users. Ask a classmate to try this command on you to see whether they see your `.plan`. In grading we will check whether you have a `.plan` file by running `finger` on each student.

Task 2: Bash shell commands (not on attu)

For each item below, **determine a single `bash` shell statement that will perform the operation(s) requested**. Each solution must be a one-line shell statement, but you may use input/output redirection operators such as `>`, `<`, `|`, and `&&`. Write these commands into your `homework4.txt` file, one per line. In your file, **write the command** that will perform the task described for each numbered item; don't write the actual output that such a command would produce. **You should perform this section on your own Linux box (not attu). CS support dislikes a surge of people needing admin intervention because of messed up permissions, and you might inadvertently open up your files for others to access :)**

1. *(Self-Discovery)* The `tar` command compresses/decompresses files in the Unix TAR ("tape archive") file format. TAR merges many files into a single archive; however, unlike ZIP, TAR does not compress the contents. Therefore most `.tar` files are then subsequently also compressed with a separate compression algorithm called GNU ZIP ("gzip"), which yields a `.tar.gz` file. (This format was used over ZIP because of patent issues.)

For this exercise, show a single command that will decompress file `hw4.tar.gz` from the current folder in "verbose" mode, so that it echoes each file that is coming out of the archive. *(Hint: You don't need `|`, `&&`, `;`, etc.)* You can download `hw4.tar.gz` on the course web site and use it for testing the other problems below.

2. Set the file `example1.txt` in the current directory so that its owner and its group can read the file. *(You don't need to change any other permissions that might currently be set on the file.)*

3. *(two parts)* (a) Set all files with extensions `.dat` and `.doc` to be read/writable (but not executable) by their owner, and to have no access by any users other than the owner. (Your command should grant these exact permissions and revoke any others.) Do this using the standard letter code arguments for granting and removing permissions.

(b) Write this same command but using the octal number code arguments to grant/remove the permissions.

4. Set all files in the current directory and all its subdirectories recursively to have a group of `admin` .

5. Give a command that would launch a text editor of your choice to edit the file `/etc/hosts` (a system configuration file that contains a mapping from internet domain names to IP addresses) as the root super-user.

6. Set all `.java` files in the current directory and all its subdirectories (and sub-sub-directories, etc.) to have read permission to all users. *(You can't achieve this simply with a `chmod` command; you need to use other commands taught recently for finding a particular group of files and processing each one as a command-line argument.)*

7. Set all files of at least 1 kilobyte in size in the current directory and all subdirs to have a group of `admin` .

8. *(Self-Discovery)* The `umask` command specifies what permissions are given by default to newly created files.

Use `umask` to set it such that when you create files, they are given read/write permission to you (the owner), but no permissions to anyone else.

Task 3: Login script, .bash_profile (on attu)

As we have discussed in class, .bash_profile is a script that runs every time you log in to a Bash shell. For this part of this assignment, you should create a `.bash_profile` file in your home directory that performs the following operations. **You also need to turn this file in.**

1. Create an alias so that when you type `attu`, you will connect to `attu.cs.washington.edu` with SSH. (This alias isn't very useful if you're testing on `attu` itself, but set it up anyway.) Set up a similar alias so that when you type `dante`, you will connect via SSH to `dante.u.washington.edu`.

2. Set it so that when you try to delete a file or overwrite a file during a move/copy operation, the user is prompted for confirmation first. *(Hint: Set each of the three operations described to run in its "interactive mode".)*

3. Lists which of your friends are online and what they are doing. Before adding this command to your `.bash_profile`, define a file named `buddylist.txt` in your home directory whose contents are the CSE user names of two or more of your friends in the department, one per line. The file can have as many lines as you like so long as it is at least two. If you don't know anybody in the department, you may use `stepp`, `rea`, or `reges`. Now add a command to your `.bash_profile` script to display which of them are logged in to the system and what they're doing, sorted by user name. For example, if your `buddylist.txt` contains:

```
stepp
rea
ln
```

Then a possible output from this command of your `.bash_profile` might be:

```
ln      pts/5    :pts/6:S.1       21:09   2:20m    0.34s   0.28s   vim poisson.ml
ln      pts/6    207.108.209.70   20:33   2:15m    0.06s   0.02s   screen
ln      pts/8    :pts/6:S.0       20:33   2:15m   11.23s   0.08s   bash ./getml.sh
stepp   pts/3    67.160.45.86     00:10   7:46     0.02s   0.04s   banner ^_^
stepp   pts/10   67.160.45.86     00:20   8:34     0.02s   0.00s   cowsay
```

(Hint: You can get this to work using commands we've already seen, but you will need to read the `man` pages.)

Note that your command should work regardless of the number of user names in `buddylist.txt`. You should not assume that `buddylist.txt` contains exactly 3 names as shown above, etc.

Recall that you can test the functionality of your `.bash_profile` by writing the following command in the shell:

```
source .bash_profile
```