CSE 390a
Lecture 4

Persistent shell settings; users/groups; permissions

slides created by Marty Stepp, modified by Jessica Miller
http://www.cs.washington.edu/390a/
Lecture summary

• Persistent settings for your bash shell

• User accounts and groups

• File permissions

• The Super User
.bash_profile and .bashrc

- Every time you login to bash, the commands in ~/.bash_profile are run
  - a . in front of a filename indicates a normally hidden file (ls –a to see)
  - you can put any common startup commands you want into this file
  - useful for setting up aliases and other settings for remote login

- Every time you launch a non-login bash terminal, the commands in ~/.bashrc are run
  - useful for setting up persistent commands for local shell usage, or when launching multiple shells
  - often, .bash_profile is configured to also run .bashrc, but not always
.bash_profile and .bashrc

• *Exercise*: Make it so that our `attu` alias from earlier becomes persistent, so that it will work every time we run a shell.

• *Exercise*: Make it so that whenever you try to delete or overwrite a file during a move/copy, you will be prompted for confirmation first.
.plan

• Another fun settings file
• Stored in your home directory
• Contains information you’d like others to be able to see
  ▪ is displayed when the finger protocol is run

• Exercise: create a quick .plan file, and make sure it works with finger
**Users**

*Unix/Linux is a multi-user operating system.*

- Every program/process is run by a user.
- Every file is owned by a user.
- Every user has a unique integer ID number (UID).

- Different users have different access permissions, allowing user to:
  - read or write a given file
  - browse the contents of a directory
  - execute a particular program
  - install new software on the system
  - change global system settings
  - ...

Groups

- **group**: A collection of users, used as a target of permissions.
  - a group can be given access to a file or resource
  - a user can belong to many groups
  - see who’s in a group using `grep <groupname> /etc/group`

- Every file has an associated group.
  - the owner of a file can grant permissions to the group

- Every group has a unique integer ID number (GID).

- Exercise: create a file, see its default group, and change it
File permissions

<table>
<thead>
<tr>
<th>command</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmod</td>
<td>change permissions for a file</td>
</tr>
<tr>
<td>umask</td>
<td>set default permissions for new files</td>
</tr>
</tbody>
</table>

- **types**: read (r), write (w), execute (x)
- **people**: owner (u), group (g), others (o)

- on Windows, .exe files are executable programs;
  on Linux, any file with x permission can be executed

- permissions are shown when you type `ls -l`

```
is it a directory?

<table>
<thead>
<tr>
<th>owner</th>
<th>group</th>
<th>others</th>
</tr>
</thead>
<tbody>
<tr>
<td>drwxrwxrwx</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Changing permissions

- **letter codes:**
  
  ```
  chmod who(+-)what filename
  ```
  ```
  chmod u+rw myfile.txt          (allow owner to read/write)
  chmod +x banner               (allow everyone to execute)
  chmod ug+rw,o-rwx grades.xls  (owner/group can read and
  note: -R for recursive       write; others nothing)
  ```

- **octal (base-8) codes:**
  
  ```
  chmod NNNN filename
  ```
  ```
  - three numbers between 0-7, for owner (u), group (g), and others (o)
  - each gets +4 to allow read, +2 for write, and +1 for execute
  ```
  ```
  chmod 600 myfile.txt          (owner can read/write (rw))
  chmod 664 grades.dat          (owner rw; group rw; other r)
  chmod 751 banner              (owner rwx; group rx; other x)
  ```
Exercises

• Change a file to grant full access (rwx) to everyone
  ▪ Now change it do deny all access (rwx) from everyone
    • !!! is it dead?
    • I own this file. Can I change the user?

• 390A website file
  ▪ How are the group permissions set up initially?
  ▪ What are the default permissions for files I upload?
    • What are the implications of that?
  ▪ How can I set up permissions so all my group members have access?
Permissions don’t travel

• Note in the previous examples that permissions are separate from the file
  ▪ If I disable read access to a file, I can still look at its permissions
  ▪ If I upload a file to a directory, its permissions will be the same as if I created a new file locally

• Takeaway: permissions, users, and groups reside on the particular machine you’re working on. If you email a file or throw it on a thumbdrive, no permissions information is attached.
  ▪ Why? Is this a gaping security hole?
• Say I have a directory structure, with lots of .txt files scattered
  ▪ I want to remove all world permissions on all of the text files
  ▪ First attempt:
    • chmod -R o-rwx *.txt
    • What happened?

▪ Try and fix this using find and xargs!
  • find -name "*.txt"
  • find -name "*.txt" | xargs chmod o-rwx
Super-user (root)

<table>
<thead>
<tr>
<th>command</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sudo</td>
<td>run a single command with root privileges (prompts for password)</td>
</tr>
<tr>
<td>su</td>
<td>start a shell with root privileges (so multiple commands can be run)</td>
</tr>
</tbody>
</table>

- **super-user**: An account used for system administration.
  - has full privileges on the system
  - usually represented as a user named root

- Most users have more limited permissions than root
  - protects system from viruses, rogue users, etc.
  - if on your own box, why ever run as a non-root user?

- Example: Install the `sun-java6-jdk` package on Ubuntu.
  
sudo apt-get install sun-java6-jdk
Playing around with power...

MAKE ME A SANDWICH.

SUDO MAKE ME A SANDWICH.

WHAT? MAKE IT YOURSELF.

OKAY.

Courtesy XKCD.com
Playing around with power...

- Create a file, remove all permissions
  - Now, login as root and change the owner and group to root
  - Bwahaha, is it a brick in a user’s directory?

- Different distributions have different approaches
  - Compare Fedora to Ubuntu in regards to sudo and su...

- Power can have dangerous consequences
  - rm * might be just what you want to get rid of everything in a local directory
  - but what if you happened to be in /bin... and you were running as root...
Wrap-up discussion

• What do you think of the permissions model in *nix?
  ▪ How does it compare to your experience of other OS’s?
  ▪ What are it’s strengths?
  ▪ Are there any limitations? Can you think of a scenario of access rights that this approach doesn’t easily facilitate?

• Additional info: ACL vs. Capabilities
  ▪ Access Control Lists
    ▪ Like what we just looked at – each file has a list of who can do what
  ▪ Capabilities
    ▪ Different approach using capabilities, or “keys”
    ▪ Principle of least privilege, keys are communicable
    ▪ Not a focus point, but more info online if you’re interested