How was the last homework assignment?

CSE 391

Users, Groups, Permissions, and more!

Joshua Ervin,
Thanks to Hunter Schafer and Brett Wortzman for the slides

Users

Linux and unix are multi-user operating systems

- Ever 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 them to

- Read or write to a given file
- Browse the contents of a directory
- Execute a particular program
- Install new software on the system
- Change global system settings
- And more!

Users

Command	Description
`whoami`	Print your username
`id`	Print user ID and group membership
`users`	List logged-in users (short)
`who`	List logged-in users (long)
`finger`	Print information about users

Groups

A group is a collection of users

- Groups can be assigned access to files or resources
- A single user may belong to multiple groups
- Check group membership with `grep <groupname> /etc/group`
- Every file has an associated group
- Every group has a unique integer ID (GID)

Groups

Command	Description
`groups`	Print group membership
`groupadd`	Create a group
`groupdel`	Delete a group
`groupmod`	Modify a group

To add users to a group, you must edit `/etc/group` as *root*. We will talk more about the *root* user later in these slides.

root

Every system contains a special user known as the *superuser* or *root*.

- Normal users are restricted in what they can read, write, execute, and install.
- The superuser has unrestricted access to the machine

There are a number of different ways to *BECOME THE SUPERUSER*

- Most systems have the *sudo* command installed, which stands for **su**per user **do**. For example, to add someone to a group, you would run `sudo vim /etc/groups`
- Additionally, you can run a superuser shell with `su`

Processes

Every user on a system has processes associated with them.

- Running a command in the shell launches a process for that command
- Process management is a major component of any operating system

Command	Description
`ps`	List processes being run
`ps -u <user>`	List processes being run by <user>
`top`	Show process statistics
`kill`	Terminate process by PID
`killall`	Terminate process by name

aliases

- Programmers are notoriously lazy, and many of the commands you use on a day to day basis tend to be verbose.
- To remedy this, you can use aliases to assign a pseudonym to a command.

Command	Description
`alias`	Assigns a pseudonym to a command

- To create an alias for a command, use the following syntax

```
alias name=command
```

- For example, to quit a shell by typing q

```
alias q=exit
```

- WARNING: DO NOT PUT SPACES ON EITHER SIDE OF THE =

.bash_profile

- Everytime you **log in** to a shell (i.e. by ssh-ing into attu or logging into your computer), the commands in `~/.bash_profile` are run.
  - It is often useful to put common startup commands (i.e. commands that you want to be run whenever you login.
- *Note*: The dot (.) that precedes the filename means the file is *hidden*. To view the file, you must run `ls -a`

.bashrc

- Everytime you launch a **non-login** shell (i.e. by launching a new terminal) the commands in `~/.bashrc` are run.
  - This is useful for setting up persistent commands.
  - In practice, your `.bash_profile` is often configured to run your `.bashrc`

$PATH

Have you ever wondered what happens when you type a command like `ls` ? How does the system know what program to run?

- Your shell looks through the $PATH variable, using the following algorithm:

```
execute(command)
    for all directories in $PATH:
        if directory contains command:
            run command
            exit
    print “command not found”
```

$PATH

- A common use case for your `.bashrc` or `.bash_profile` is adding directories to your path.
- To *prepend* a directory to your `$PATH`

```
export PATH=/your/new/directory:$PATH
```

- To *append* a directory to your `$PATH`

```
export PATH=$PATH:/your/new/directory
```

Poll Everywhere

Think

1 minute

pollev.com/cse391

Suppose we have the following executable files and `$PATH` variable

```
PATH=/usr/home:/usr/bin:/usr/sbin
```

`/usr/bin/hello.sh`

```
echo “howdy partner”
```

`/usr/sbin/hello.sh`

```
echo “yeehaw”
```

What would be the output of the following command?

```
> hello.sh
```

Poll Everywhere

Pair

1 minute

pollev.com/cse163

Suppose we have the following executable files and `$PATH` variable

```
PATH=/usr/home:/usr/bin:/usr/sbin
```

`/usr/bin/hello.sh`

```
echo “howdy partner”
```

`/usr/sbin/hello.sh`

```
echo “yeehaw”
```

What would be the output of the following command?

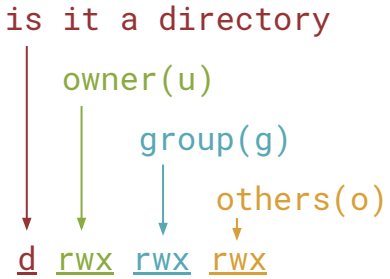
```
> hello.sh
```

File Permissions

Since multiple users share the same system, files and directories are protected by certain permissions.

- Permissions for each file are defined by the type of person trying to do something and the operation that they are trying to run.
- The types of people are: owner(u), group(g), and others(o)
- The types of operations are: read(r), write(w) and execute(x)

Permissions are shown by typing `ls -l`



File Permissions

People: each user fits into exactly one of the following categories:

- `owner(u)` - the person who created the file. This can be changed using chown
- `group(g)` - the group associated with each file
- `others(o)` - people who are neither the owner of the file nor a member of the group associated with the file

Permissions:

- `read(r)` - allows files to be open and read
- `write(w)` - allows contents of a file to be modified
- `execute(x)` - allows the file to be run

Changing File Permissions

The first method of changing file permissions is using letter codes. It uses the following general syntax:

```
chmod who(+-)what filename
```

- allow group members to execute:

```
chmod g+x file.txt
```

- allow everyone to execute:

```
chmod +x file.sh
```

- owners/group read/write, others nothing:

```
chmod ug+rw,o-rwx file.sh
```

- NOTE: THERE IS NO SPACE AFTER THE COMMA

Changing File Permissions

The second method of changing file permissions is using octal. It uses the following general syntax (N is 0-7):

```
chmod NNN filename
```

- Each digit corresponds to owner(u), group(g), and others(o) in that order.
- Each gets +4 for read, +2 for write, and +1 for execute
- Owners can read/write, everyone else gets nothing

```
chmod 600 file.txt
```

- Everyone can read, write, and execute

```
chmod 777 file.txt
```

Poll Everywhere

Think

1 minute

pollev.com/cse391

Using letter codes, what would be the command to allow the owner and group to read, write, and execute and others to read?

Poll Everywhere

Think

1 minute

pollev.com/cse391

Using octal codes, what would be the command to allow the owner and group to read, write, and execute and others to read?

Default File Permissions

Finally, there is a program called umask which sets the default permissions for newly created files. The syntax is as follows:

```
umask [options] [mask]
```

- Calling umask without any arguments will print out the current mask.
- Calling umask similarly to chmod with octal will *remove* those permissions for newly created files.
- The following will give the owner full privileges and all others read/execute privileges only.

```
umask 0022
```

- Note that the leading zero for umask is intended to signify that it is an octal number (base 8).

Directory Permissions

We noted earlier that the first digit of a file's permissions determines whether or not that file is a directory. Directories have the same permissions as files, although the behavior is a little different:

- Read - determines whether or not the user can view the contents of the directory (i.e. can they run `ls` on that directory)
- Write - determines whether the the user can add or delete files or directories.
- Execute - determines whether the user can enter into that directory.

Messaging

You can send messages to other users on attu using the `write` command

```
write <user>
```

You can prevent other users from sending you messages using the `mesg` command

```
mesg (y|n)
```

For example, if I wanted to send a message to Zorah, I would write

```
write zorahf
```