Lecture 4

Thursday, September 24, 2015 2:29 PM

ls -l: type-permissions links to file userid group file size last modified date filename

Permissions: ___ ___ ___ 3 groups

1. User bits - owner's permissions
2. Group permissions
3. Other permissions

- If you are the user, you do not have the same permissions as the group permission settings
- e.g.) User can read/write (r/w), group can execute (x)
  - User cannot execute (x)

	Ordinary files	Directory
r	• read contents • e.g.) cat	• see contents of directory • files inside directory • e.g.) globbing patterns, run ls, tab completion
w	• write contents	• add or remove files • modify contents of directory
x	• run file as program • if file is program	• navigate into directory • e.g.) cd [directory name]

- Permissions of the directory do not dictate the permissions of the files inside of the directory

Changing permissions:

- Owner is the only one with the permission to change file permissions
  - Exception: (administrator) can do this as well
- **chmod**: <mode> file (can use globbing patterns)
  -

Mode:	Ownership class	Operator	Permission
	• u - user • g - group • o - other	• add + • remove - • set exactly =	• r • w • x

	• a - all		

- e.g.) give others read permission:
  - $> chmod o+r file
- e.g.) revoke exec permission from group
  - $> chmod g-x file
- e.g.) make everyone's permission rw
  - $> chmod a=rw file
- e.g.) change mode using bits
  - $> chmod 743 file
    - 743 = 111 100 011
    - rwx r-- -wx
    - Common one is 644

Shell variables:

- $> x=1 **no spaces**
- echo $x or ${x}
- Special variables:
  - ${PATH}
  - Paths to search for programs
  - e.g.) ls -> shell searches all directories in ${PATH} and find program

Shell scripts:

- Text file that contains linux commands
- Executed as a program

- e.g.)

1. Specify what type of script it is
   - #!/bin/bash - *shebang line* needs to be there
2. Write commands

- e.g.)
  - Pwd
  - whoami
  - Date

3. Run script
   - The script's directory must be on the ${PATH}
     OR
   - ./<script name>
   - Make sure execute bit is set

- Make sure execute bit is set

Command Line Arguments (scripting):

- Within a script, argument 1 is accessible through the variable '$1'
- Argument2 is accessible through '$2' … argument n is accessible through '$n'
- ./script arg1 arg2
  - Within our script, $1 = arg1, $2 = arg2, $3 = empty string
  - $0 is the name of the script
  - **shift** = $1 discarded, $2 takes $1's place
    - Like a queue
- e.g.) check whether a given word is in the dictionary
  - ./isWordInDict word
    #!/bin/bash
    egrep "^${1}$" /usr/share/dictionary

```
#!/bin/bash
egrep "^$1$" /usr/share/dict/words
```

- e.g.) a good password is not in the dictionary
  - Answer whether a given word is a good password
  - ./goodPassword word
    #!/bin/bash
    egrep "^${!}$" /usr/share/dictionary > /dev/null
    if [$? -eq 0]; then - **MUST have spaces where indicated**
      echo "Bad password"
  - **/dev/null**
    - "black hole" to throw away output

```
#!/bin/bash
# Answers whether a word is in the dictionary (and therefore not a good
#  password)

egrep "^$1$" /usr/share/dict/words > /dev/null

if [ $? -eq 0 ]; then
  echo Not a good password
else
  echo Maybe a good password
fi
```

Notes:

- All programs return a status code
  - 0 for success
  - Non 0 for failure
- egrep has status code 0 if pattern is matched
  - 1 if never matched

- Status code is available in **"$?"**
  - From last command
- **if** is actually a separate program that evaluates conditional statements
  - Close if statements with **fi**

- If [cond]; then

  _____

  elif [cond]; then

  _____

  else

  _____

  Fi
- Useful conditions:
  - -e file = checks whether a file exists
    - Sets status code 0 if found
    - 1 not found
  - -a **and**
  - -o **or**
  - -eq, -ne **integer comparisons**
  - =, != **string comparisons**