EASTERN WASHINGTON UNIVERSITY
start something big

Unix I/O Redirection and Piping

Computer Science Department
Eastern Washington University
Yun Tian (Tony) Ph.D.

Recall Last Class

- Now you know using metacharacters in your command, such as *, ?, [], [^….].
- Using !! to run last command you did.
- Using !cd to run your history command that starts with cd
  - !echo run your history command that starts with echo.
- PATH and .bashrc file in your home.

Outline for Today

- Shell Sequence
- Redirection
- Pipe

PATH and Shell Sequence

- When you type in a command, shell goes through quite a complicated sequence of operations to process your request.
  1. Check aliases
  2. Parameter expansion, substitution, quotes removal.
  3. Shell function
     - function hello() { echo "Hello from function()" ; }

PATH and Shell Sequence

4. Buildin command
5. Hash tables
   - Cached PATH entry for a command that was previously executed.
6. PATH variable

- If everything fails, you see “Command not found”.
  - You see it when you type in **flykite** in shell

PATH and Shell Sequence

- PATH environment variable specifies a list of directories the shell searches for the commands.
  - why most commands(executable) are placed in /bin or /usr/bin?
    - /bin and /usr/bin by default is added in PATH.
    - Shell will search commands in these directories listed in PATH.
      - Shell have to locate the command and execute it.
    - You can find most of command (executable file) there.

      ls –alh /bin

Find Where is the Program

- **which <command>** searches the directories in $PATH.
  - locate a program file in the user's path.
- **whereis <command name>**
- **locate <pattern to search>**
  - You use locate or whereis to find a command such as you know it is there somewhere but not in your path.

Find Where is the Program

- **Find**
  - Search files using name, size, permission, time and other criteria.
  - Like search files in Windows.
  - More general and looks for all sorts of things . (coming more later)

Standard I/O Steams

- Every Unix program has three streams opened for it, one for input, one for output, and one for printing diagnostic or error messages.
  - The input stream is referred to as "standard input";
  - the output stream is referred to as "standard output";
  - the error stream is referred to as "standard error".
- stdin, stdout and stderr

Standard I/O Steams

- Every device or stream is considered a file in Unix.
  - The integer file descriptors associated with the streams stdin, stdout, and stderr are 0, 1, and 2, respectively.
- The standard output -- the screen on your terminal by default.

Standard I/O Steams

- The standard input, typically the keyboard.
  - When a command reads the standard input, it usually keeps reading text until you type Control + d. (This is equivalent to EOF file flag)
- You redirect these streams -- to a file, or even another command -- with *redirection.*

Standard I/O Steams

- You redirect these streams -- to a file, or even another command -- with *redirection.*

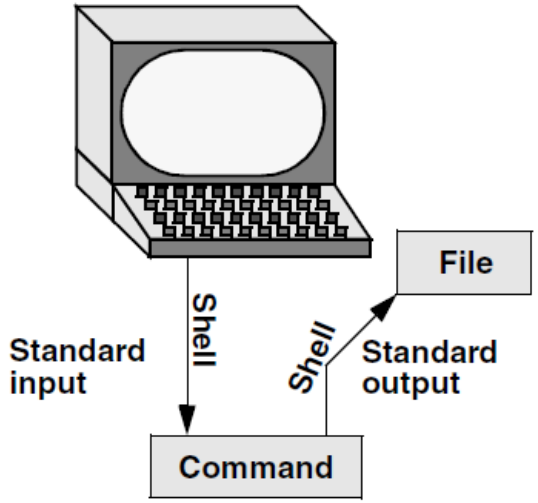

Figure 5-6 Redirecting standard output

http://itknowledgeexchange.techtarget.com/bookworm/book-excerpt-a-practical-guide-to-linux/

Redirecting Input to a File

- To redirect standard input to a file, use the < operator.
  - You may want a file to be the input for a command that normally wouldn't accept a file as an option.
  - wc –l < hw1.c
  - Command **wc** that means word count, takes a file hw1.c as input using I/O redirection.

Redirecting Input to a File

- To redirect standard input to a file, use the < operator.
  - 'wc –l < hw1.c' counts how many lines in hw1.c
- Another example:

  mail tony@good.org < todo.txt

  Send eMail to adress tony@good.org. The mail content has already been saved in file todo.txt.

Redirecting Output to a File

- To redirect standard output to a file, use the > operator.
  - **ls –lah > details**
  - **who > user_list**
    - Show who is logged in, and save the results in user_list file.
- If you redirect standard output to *an **existing file***, it will overwrite the file, unless you use the `>>' operator.
  - '>> ' appends the standard output to the contents of the existing file.

Redirecting Output to a File

- Example of using the `>>' operator.
  - cal 10 2012 >> existing_calendar
    - Cal 10 2012 show calender for October 2012.
    - The command above appends its output to the existing file, existing_calendar.
  - date >> existing_calendar
    - Appends the current date information at the end of existing_calendar file.

Redirecting with noclobber

- It happens when you accidentally overwrite a file by using redirection.
  - ls –l *.c > output.txt
  - You overwrite output.txt.
  - It is useful when you mistake > for >>.
- You need to set noclobber variable.
  - It can keep you from accidentally destroying your existing files.

Redirecting with noclobber

- **set -o noclobber**
  - If you now try to redirection to already existing output.txt by using ls –l > output.txt,
  - You got a message 'can not overwrite existing file'
- Permanently add set –o noclobber to your .bashrc file.
  - echo 'set -o noclobber' >> ~.bashrc

Redirecting with noclobber

- If you like to temporary turn off noclobber for one single operation,
  - After set –o noclobber, you like to make exception for individual command.
  - Use >| to force the file to be overwritten
    - **ls –l >| existing_list**
- **set +o noclobber**
  - Turn off noclobber, now you can overwrite existing files in redirection.

Redirecting Error Messages to a File

- To redirect the standard error stream to a file, use the > operator preceded by a '2'.
  - **ls –l > outfile 2>errfile**
  - 2 is the file descriptor for standard error stream.
  - Standard output is redirected to outfile
  - Standard error message is redirected to errfile.
- To redirect *both* standard output and standard error to the same file, use `&>' instead.

Pipe

- Piping is when you connect the standard output of one command to the standard input of another.
  - **ls –l | wc –l**
    - count how many files and folder in current directory
  - **who | grep 'tony'**
    - Check whether tony is logged in.
    - First run who to list all logged in users, then search 'tony' in that list.

Pipe

- Piping is to connect two or more commands, output of first command becomes the input of the second one.
  - Differs from I/O redirection.
  - I/O redirection redirects input/output to text files.
- Redirection with Piping
  - **who | grep 'tony' > user_file**
  - **col –b < badfile > goodone**

Summary

- Shell sequence, when input a command, how the shell search many places.
- I/O redirection, >, <, >> operators
- **set -o noclobber** to prevent accidental overwirte.
- **Piping**
  - **ls –l | wc –l**
- **whereis** and **locate** to find a program

Next Class

- More advanced commands
  - grep and find