System Programming

Shell Operators

Three Standard Files

- stdin – standard input
  - input character stream
  - Defaults to keyboard
- stdout – standard output
  - output character stream
  - Defaults to terminal
- stderr – standard error
  - receives error messages
  - Defaults to terminal

Redirecting stdout

- Instead of sending stdout to the terminal, you can tell a program to write to a file
- > *filename* : redirect stdout to a file
  - file is created if it does not exist
  - file is zeroed if it does
- >> *filename* : append stdout to an existing file
- Examples:
  - `man ls > ls_help.txt`
  - `echo $PWD > current_directory`
  - `cat file1 >> file2`

Redirecting stdin

- Instead of reading from a terminal, you can tell a program to read from a file
- < *filename* : redirect stdin from an existing file
- << *word* : redirect stdin from lines that follow up until a line containing just *word*
  - Parameter substitution, back-quoted commands and backslash character on the lines are interpreted
- Examples:
  - `mail user@domain.com < message.txt`
  - `at 3am < cmds  or at 0945 < cmds`
  - `sort < friends > sorted_friends`
  - `cat << end`

Standard File Descriptors

- A file can be associated with a file descriptor
- The shell associates three standard files with three standard file descriptors for every command respectively
  - 0 : standard input (STDIN)
  - 1 : standard output (STDOUT)
  - 2 : standard error (STDERR)
- Standart descriptors are associated with the user's terminal, but they can also be redirected into other files.

File Descriptor Creation

- To open a file for writing, use one of these

  exec *n>filename*

  exec *n>>filename*

  where
  - *n* is an integer, and *filename* is the name of the file opened for writing.
  - The first form overwrites the specified *filename* if it exists.
  - The second form appends to the specified *filename*.
- To open a file for reading, use

  exec *n<filename*
- To open a file for both reading and writing, use

  exec *n<>filename*

Redirection with File Descriptors

- To redirect standard output to the file associated with file descriptor *n*, use

  command >&n
- To redirect standard input from the file associated with file descriptor *n*, use

  command <&n
- exec n>&–: output file descriptor n is closed
  - exec 1>&–, exec >&– : standard output is closed
- exec n<&–: input file descriptor n is closed
  - exec 0<&– , exec <&– : standard input is closed

Redirection with File Descriptors

- Writing to a file

```
exec 4> file                # Open "file", assigning fd 4 to it.
ls >&4                      # Write ls output to "file"
```

- Reading from a file

```
exec 5< file                # Open "file", assigning fd 4 to it.
wc <&5                      # Read input from “file"
```

- Writing at a specified place in a file

```
echo 1234567890 > file      # Write string to "file".
exec 3<> file               # Open "file", assigning fd 3 to it.
read -n 4 <&3               # Read only 4 characters.
echo -n . >&3               # Write a decimal point there.
exec 3>&-                   # Close fd 3.
cat file                    # ==> 1234.67890
```

General Input/Output Redirection

- The standard output or input redirection is explicitly indicated in the general form as

  command 1> *file*

  command 1>> *file*

  command 0< *file*

- The basic syntax to redirect STDOUT and STDERR to separate files is

  command 1> *fileA* 2> *fileB*

  - the STDOUT of the specified *command* is redirected to *fileA*, and the STDERR (error messages) is redirected to *fileB*.

Redirection To Separate Files

- The append operator can be also used.

  command >> fileA 2> fileB

  command > fileA 2>> fileB

  command >> fileA 2>> fileB

  - The first form appends STDOUT to *fileA* and redirects STDERR to *fileB*.
  - The second form redirects STDOUT to *fileA* and appends STDERR to *fileB*.
  - The third form appends STDOUT to *fileA* and appends STDERR to *fileB*.

Redirection To A Single File

- The basic syntax to redirect or append STDOUT and STDERR to the same file is

  command > file 2>&1 or *command &> file*

  command >> file 2>&1

  - STDOUT (file descriptor 1) and STDERR (file descriptor 2) are redirected or appended into the specified *file*.

  command 2>&1 >> *file* *(compare this to above)*

- m>&n: file descriptors m is redirected to n
  - >&n: standart output is redirected to file descriptor n
- Example

```
rm -rf /tmp/my_tmp_dir > /dev/null 2>&1
rdate ntp.nasa.gov >> /var/log/rdate.log 2>&1
if [ ! -f $FILE ]; then echo "$FILE is not a file" >&2; fi
```

Pipes

- Pipe (|) : connect stdout of one command to stdin of another
- Examples
  - `ls –la | less`
  - `ls –al | wc`
  - `ls-al | sort +4r`
  - `cat file | wc`
  - `man bash | grep "history“`
  - `ps aux | grep user1 | wc –l`
  - `who | sort > current_users`

Processes

- Run more than one program at a time
- Separate commands with a semicolon (;)

  ```
  date ; who
  ```
- Run more than one program simultaneously
- Use an ampersand (&) at the end of a command

  ```
  ls –al & wc *
  ```

Filters

- **Filter** – a program that takes input and transforms it in some way
  - *`wc`* – gives a count of lines/words/characters
  - *`grep`* – searches for lines with a given pattern
    - `grep <pattern> <filename>` (pattern can be RE)
  - *`sort`* – sorts lines alphabetically or numerically
    - `sort -r` : reverse normal order of sorting
    - `sort -n` : sort in numeric order
    - `sort +2n` : sort items in the second column
  - `cut` – select parts of each line to send to stdout
    - `cut -c1-5`: select the first 5 characters of each line
    - `cut -c1,5`: select the first and fifth chars of each line
    - `cut -d: -f1,3 /etc/passwd` : map user names to IDs

Filters (cont.)

- *`head`* – display first few lines of files
  - `head -n <filename>`   `n:` an integer
- *`tail`* – display the last part of a file
  - `tail -n <filename>` : the last n lines
  - `tail +n <filename>` : lines after the n*th* line
- *`diff`* – report on all the lines that are different
- *`cmp`* – find the first place where two files differ
  - `<diff/cmp> <file1> <file2>`
- *od* – display octal representation of a file
  - e.g. `od –c` : visual representation of all bytes
- `ls –lt` : list file in time order
- *`crypt`* `– encode or decode a file`
  - e.g. `crypt key < clear.file > encrypted.file`

Filters (cont.)

- *`tr`* – transliterate the characters in its input
  - `tr "[:lower:]" "[:upper:]" < <file>`
    - map lower case to upper
- *`uniq`* – report or filter out repeated lines in a file
  - `uniq –d <file>` - display lines repeated in <file>
  - `uniq –u <file>` - display lines not repeated in <file>
  - `uniq –c <file>` - display with number of lines repeated
- *`pr`* – print files in various forms
  - *`ls -a | pr -n -h $(pwd)`*
    - *`print`* a numbered list of all files in the current directory
- What does the following command do?

  -
    ```
    cat * | tr -sc A-Za-z '\012' | sort
        | uniq –c | sort –n | tail | pr -5 –t
    ```

More Commands: Communication

- `talk` – interactive chat with another user
  - e.g. `talk smith pts/2`
- `write` – send message to another user
  - e.g. `write smith pts/2`
  - `mesg [n|y]`– permit/deny messages
- `mail`, `pine` – text- based email program
- `ftp` , `sftp` – text-based FTP program
- `telnet` , `ssh` – connect to other machines directly
- `lynx` – text-based web browser

More commands: Processes

- `ps` – list current processes
- *`top`* – dynamic display of system's utilization by processes
- `kill` – terminate a process (default: SIGTERM)
  - `kill -9 <pid>        (sending SIGKILL signal)`
- *`time`* – keep timing information for a process
  - `time ls      (displaying real/user/sys time)`
- `wait` – waiting for all procs initiated with &
- `nohup` – keep command running after logging out
- `nice` – keep command running with lower priority
  - `nohup/nice <command> &`

More commands: File system

- *`file`* – determine file type
  - `file /bin/ed`

    ```
    /bin/ed:     pure executable
    ```
- A runnable program is marked by a binary "magic number" at its beginning.
  - *`od /bin/ed`*

    ```
    0000000 077505 046106 000402 000400 000000 ...
    ```
- `du` – tell how much disc space is consumed
  - `du <file/directory>` (in terms of disc blocks)
- `df` – report space on the mounted file subsystems
  - `df -k` (in terms of disc blocks) (a block = 512 or 1024 bytes)