Standard input/output

Tutor: Lưu Thanh Trà
Email: lttra@hoasen.edu.vn

Outline

- General form of commands
- Writing output
- Redirect output
- Standard input
-

General form

- Do_something < input_file > output_file

Writing output – echo

- $ echo Please wait.

  Please wait.
- $ echo this was very widely spaced

  this was very widely spaced
- $ echo "this was very widely spaced"

  this was very widely spaced
- $ echo 'this was very widely spaced'

  this was very widely spaced

printf

- Ex:
  - printf '%10.10s = %4.2f\n' 'GigaHerz' 1.92735
  - GigaHerz = 1.93
- String
  - %10.5s: max – min number of char
- Real number
  - %4.2f : max number of digits – number of digit after the decimal point

Writing output without the newline

```
$ printf "%s %s" next prompt
next prompt$

$ echo -n prompt
prompt$
```

Outline

- General form of commands
- Writing output
- **Redirect output**
- Standard input
-

Saving output with “>” “>>”

- Example
- “>” : clobber output

  $ echo fill it up

  fill it up

  $ echo fill it up **>** file.txt

  $ echo some more data > ../../over.here
- “>>” : append output
- Throw output away
  - find / -name myfile –print 2 > /dev/null
- Prevent a file from overwritten
  - set +o noclober

- Several commands output
  - {pwd;ls; cd ../elsewhere;pwd; ls} < /file1
  - (pwd;ls; cd ../elsewhere;pwd; ls) < /file1
- Note: for (), each command is implemented in a sub-shell, when the command is done, we return to the original shell

Showing the end/beginning of a file

- Example
  - the end of a file
    - tail –n +3 file_name : skip two first lines of the file
    - tail –2 file_name: show 2 line from the bottom
  - the head of a file
    - head –n 2 file_name: show 2 first lines
    - head –n -2 file_name: show all but 2 lines from the bottom

Standard output & Standard error

- 1 : standard output
- 2 : standard error
- 0 : standard input
- Example

```
$ myprogram 1> messages.out  2> message.err
$ myprogram > messages.out  2> message.err
$ both > outfile 2>&1
$ both >& outfile
$ both &> outfile
```

Saving a copy of output

- Tee command allows users to print the output on the screen and copy it to a file
  - $ cat my* | tr 'a-z' 'A-Z' | uniq | awk -f transform.awk | wc
  - $ ... uniq | tee /tmp/x.x | awk -f transform.awk ...
- Using tee to print output on the screen instead of waiting the process completes
  - find / -name '*.c' -print > /tmp/all.my.sources
  - find / -name '*.c' -print | tee /tmp/all.my.sources
  - find / -name '*.c' -print 2>&1 | tee /tmp/all.my.sources
- For commands which do not accept the standard output
  - find . –name ‘*.class’ | rm => not working !
  - rm $(find . –name ‘*class’)

Swapping stderr & stdout

- $ ./myscript 3>&1 1>&2 2>&3
  - fd 3 is a duplicate of 1
  - 1 is equal to 2
  - 2 is assigned to 3
- $ ./myscript 3>&1 1>stdout.logfile 2>&3- | tee -a stderr.logfile
  - “-” after 2>&3 is to close the fd 3

Outline

- General form of commands
- Writing output
- Redirect output
- **Standard input**

Standard input

- Example
  - wc < my_file
  - wc my_file
- Get user input
  - read
  - read – p "answer me this " ANSWER
  - read PRE MID POST

Keep data in a script

- Example

```
$grep $1 <<EOF
mike x.123
joe x.234
sue x.555
pete x.818
sara x.822
bill x.919
EOF
$
```

- We can turn off the shell scripting features using \EOF

Review of function

1.
```
#!/bin/sh
# A simple script with a function...

add_a_user()
{
  USER=$1
  PASSWORD=$2
  shift; shift;
  # Having shifted twice, the rest is now comments ...
  COMMENTS=$@
  echo "Adding user $USER ..."
  echo useradd -c "$COMMENTS" $USER
  echo passwd $USER $PASSWORD
  echo "Added user $USER ($COMMENTS) with pass $PASSWORD"
}
```

2.

```
###
# Main body of script starts here
###
echo "Start of script..."
add_a_user bob letmein Bob Holness the presenter
add_a_user fred badpassword Fred Durst the singer
add_a_user bilko worsepassword Sgt. Bilko the role model
echo "End of script...“
```

3. Note: $1=bob $2=letmein $3=Bob
$4=Holness $5=the $6=presenter

```
function choose {
  local default="$1"
  local prompt="$2"
  local choice_yes="$3"
  local choice_no="$4"
  local answer
  read -p "$prompt" answer
  [ -z "$answer" ] && answer="$default"
  case "$answer" in
  [yY1] ) exec "$choice_yes"
  # error check
  ;;
  [nN0] ) exec "$choice_no“
  ;;
  * ) printf "%b" "Unexpected answer '$answer'!" >&2 ;;
  esac
  }
```