Writing Shell Scripts – part 3

CSE 2031
Fall 2010

Changing Values of Positional Parameters

- Positional parameters **`$1, $2,`** … normally store command line arguments.
- Their values can be changed using **`set`** command , for example, **`set `date``**
- The new values are the output of **`date`** command.

Example

```
% cat setparm
#!/bin/sh
echo "Hello, $1. You entered $# command line argument(s). Today's date is ..."
date
set `date`
echo There are now $# positional parameters.  The new parameters are ...
echo \$1 = $1, \$2 = $2, \$3 = $3, \$4 = $4, \$5 = $5, \$6 = $6.

% setparm Amy Tony
Hello, Amy. You entered 2 command line argument(s). Today's date is ...
Sat Nov 27 11:55:52 EST 2010
There are now 6 positional parameters. The new parameters are ...
$1 = Sat, $2 = Nov, $3 = 27, $4 = 11:55:52, $5 = EST, $6 = 2010.
```

3

Debugging Tools

`sh –v myscript`

- **`v`**: verbose
- displays each command it finds in the script as it encounters it (before substitution).
- allows you to find which particular line in your code has the syntax error. Displaying will stop at this point and the script exits.

- Example:

`sh -v setparm Amy`

4

Debugging Tools (2)

`sh –x myscript`

- **`x`**: execute
- similar to **`–v`**, but displays a command only when it is executed (before execution but after substitution).
- useful for debugging control structures (**if**, **case**, loops).
  - if no control structures then **`x`** and **`v`** display the whole program.
- puts a plus sign (+) in front of any command that gets processed (easier to read than **`–v`**).
- Examples:

```
sh -x setparm Amy
sh -x chkex ghost   # compare with -v
```

5

Debugging Tools (3)

`sh –xv myscript`

- Both options may be used at the same time.

- To check variable substitutions.
- Example:

```
sh -xv setparm Amy
```

- To view the whole program and its execution.
- Example:

```
sh -xv chkex ghost
```

6

Debugging Tools (4)

`sh -n myscript`

- Reads the commands but does NOT execute them.
- Useful for “compiling” the script to detect syntax errors.
- Example uses:
  - a good working script will modify/delete files.
  - interactive input from user is required.
  - very long scripts.

7

I/O Redirection of Control Structures

- Control structures can be treated as any other command with respect to I/O redirection.
- Example: **`myscan`** reads all the regular files from current directory downwards, sorts them and searches each file for a pattern entered via positional parameter **`$1`**; if pattern exists, the file name is stored in file **`$1.out`**.

```
% cat myscan
#!/bin/sh
find . -type f | sort | while read File
do
   grep -l $1 $File
done > $1.out
```

8

Shell Functions

- Similar to shell scripts.
- Stored in shell where it is defined (instead of in a file).
- Executed within **sh**
  - no child process spawned
- Syntax:

```
function_name()
{
  commands
}
```

- Allows structured shell scripts

9

Example

```
#!/bin/sh
# function to sample how many users are logged on
log()
{
  echo "Users logged on:" >> users
  date >> users
  who >> users
  echo "\n\n" >> users
}

# taking first sample
log

# taking second sample (30 min. later)
sleep 1800
log
```

10

Shell Functions (2)

- Make sure a function does not call itself causing an endless loop.

```
% cat makeit
#!/bin/sh
…
sort()
{
   sort $* | more
}
…
```

- Should be written:

```
% cat makeit
#!/bin/sh
…
sort()
{
   /bin/sort $* | more
}
…
```

11

Environment and Shell Variables

- Standard UNIX variables are divided into 2 categories: shell variables and environment variables.
- **Shell variables**: apply only to the current instance of the shell; used to set short-term working conditions.
  - displayed using `'set'` command.
- **Environment variables**: set at login and are valid for the duration of the session.
  - displayed using `'env'` command.
- By convention, environment variables have UPPER CASE and shell variables have lower case names.

12

Environment and Shell Variables (2)

- In general, environment and shell variables that have "the same" name (apart from the case) are distinct and independent, except for possibly having the same initial values.
- Exceptions:
- When **`home`**, **`user`** and **`term`** are changed, **`HOME`**, **`USER`** and **`TERM`** receive the same values.
- But changing **`HOME`**, **`USER`** or **`TERM`** does not affect **`home`**, **`user`** or **`term`**.
- Changing **`PATH`** causes **`path`** to be changed **and vice versa**.

13

Variable `path`

- PATH and path specify directories to search for commands and programs.

```
cd              # current dir is home dir
chex report     # chex is in 2031/Lect, so failed
echo $path
set path=($path 2031/Lect)
echo $path
chex report     # successful
ls -l report
```

- To add a path permanently, add the line to your **`.cshrc`** (or **`.bashrc`**) file after the list of other commands.

```
set path=($path .)    # avoid typing ./a.out
```

14

set Command Summary

- Displays shell variables
- Set variables
- Changing command-line arguments
  - **`set `date``**

15

`break` and **`continue`**

- Interrupt loops (**`for`**, **`while`**, **`until`**)
- **`break`** transfers control immediately to the statement after the nearest **`done`** statement
  - terminates execution of the current loop
- **`continue`** transfers control immediately to the nearest **`done`** statement
  - brings execution back to the top of the loop
- Same effects as in C.

16

`break` and `continue` Example

```
#!/bin/sh
while true
do
echo “Entering ‘while’ loop ...”
echo “Choose 1 to exit loop.”
echo “Choose 2 to go to top of loop.”
echo -n “Enter choice: ”
read choice
if test $choice = 1
then
  break
fi
echo “Bypassing ‘break’.”

if test $choice = 2
then
  continue
fi

echo “Bypassing ‘continue’.”
done

echo “Exit ‘while’ loop.”
```

17

`$*` versus `$@`

- `$*` and `$@` are identical when not quoted: expand into the arguments; blanks in arguments result in multiple arguments.
- They are different when double-quoted:
- `"$@"` each argument is quoted as a separate string.
- `"$*"` all arguments are quoted as a single string.

18

$* versus $@ Example

```
% cat displayargs
#!/bin/sh
echo All the arguments are "$@".
countargs "$@"
echo All the arguments are "$*".
countargs "$*"

% cat countargs
#!/bin/sh
echo Number of arguments to countargs = $#

% sh -xv displayargs Mary Amy Tony
```

19

Next time

- C again:
  - Program structure (cont.)
  - File I/O
  - makefile
- Review
- Reading for this lecture:
  - 3.6 to 3.8, UNIX textbook
  - Posted tutorial on standard UNIX variables
- See Chapter 5, UNIX textbook for more examples.

20