CS 5142
Scripting Languages

9/23/2013

Bash Scripting

Outline

- Bash

About Bash

- Bash is one of many Unix shells
- Unix shell = interactive command line interface
  - Used to manipulate files and control processes
  - Shell script = sequence of shell commands stored in a file and used like a program
  - As users wrote more elaborate shell scripts, shells grew into early scripting languages
- History / naming
  - Original Thompson shell (Ken Thompson 1969)
  - Replaced by Bourne shell (Stephen Bourne 1977)
  - Most shells today are Bourne-compatible
  - Bash = Bourne-Again SHell (Brian Fox 1987)
  - Bash is default on Linux, Mac OS X, and Cygwin

How to Write + Run Code

- Read-eval-print loop: **`bash`**
- Run script from file: **`bash`** *file* **`.sh`**
- Run script stand-alone: **`#!/bin/bash`**
- One-liner: **`bash -c '`***command*'
- Runs automatically:

  `.bashrc`, **`.bash_profile`**

Example

```
#!/bin/bash
USAGE=$(cat <<EOF
Usage: compileall [options]
Options:
  -v | -verbose   Show executed commands.
  -p | -pretend   Don''t actually compile, only show commands.
EOF
)
VERBOSE=false PRETEND=false
while [ 0 -lt $# ]; do
  case "$1" in
    -v | -verbose)  VERBOSE=true;;
    -p | -pretend)  PRETEND=true VERBOSE=true;;
    *)              echo "Unknown option $1."; echo "$USAGE"; exit 1;;
  esac
  shift
done
for f in *.c; do
  COMMAND="gcc -o ${f%.c}.exe $f"
  if [[ $VERBOSE == true || $PRETEND == true ]]; then echo $COMMAND; fi
  if [[ $PRETEND == false ]]; then $COMMAND; fi
done
```

Lexical Peculiarities

- Single-line comments: `#`…
- Commands terminated by one of the control operators (`;`, `&`, `&&`, `||`, *newline*)
- Expansion (see next slide)
  - Sigil `$` for variable use, but not when left-hand-side of assignment or declaration
  - Don't need quotes around strings
  - Don't need commas between parameters
- Heredocs

Expansion

Brace expansion	`a{b,c}d`	⇒	`abd acd`
Tilde expansion	`~`	⇒	`/home/hirzel`
Left-to-right: Parameter and variable expansion	`$foo`	⇒	Like interpolation in Perl, value of `$foo`
Left-to-right: Arithmetic expansion	`$((2+2))`	⇒	`4`
Left-to-right: Command substitution	`$(…)`, `` `…` ``	⇒	Output of executing
Word splitting	Split unless grouped by `"…"` or `'…'`		
Pathname expansion	`foo.*`	⇒	`foo.ppt` `foo.pdf`
Quote removal	`'a\'b\\'`	⇒	`a'b\`
Command execution	Treat first word as function, builtin, or program, rest as parameters		

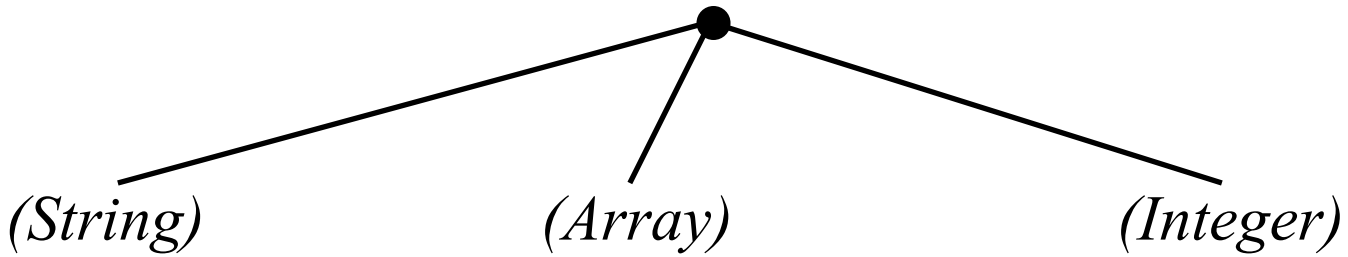
Structure of a Bash Application

- *command* ::= (*id*=*expr*)* *wordsAndRedirections controlOp*
- *pipeline* ::= *command* (| *command*)*
- *list* ::= *pipeline* ((;|&|&&|||) *pipeline*)* (;|&|*newline*)
- *compoundCommand* ::=
  - (*list*) -- execute in subshell
  - | { *list* ; } -- execute in current shell environment
  - | ((*expr*)) -- arithmetic; exit status 0 iff expression value !=0
  - | [[*expr*]] -- conditional, with (…), !, &&, ||
  - | *controlStatement*
- Include file (run commands in current environment):
  - (**source** | **.**) *filename args*
- Exit status
  - Every command has an exit status
  - 0 treated as true by if, while, conditionals

Control Statements

Conditional		**`if`** …**`;`** **`then`** …**`;`** **`elif`** …**`;`** **`then`** …**`;`** **`else`** …**`;`** **`fi`** **`case`** *id* **`in`** *pat*`|`…`)` …`;;` … **`esac`**
Loop	Fixed	**`select`** *id* [**`in`** …]**`;`** **`do`** …**`;`** **`done`** **`for`** *id* [**`in`** …]**`;`** **`do`** …**`;`** **`done`** **`for ((`**…**`;`** …**`;`** …**`));`** **`do`** …**`;`** **`done`**
	Inde-finite	**`while`** …**`;`** **`do`** …**`;`** **`done`** **`until`** …**`;`** **`do`** …**`;`** **`done`**
Unstructured control		**`break`** **`continue`** [*n*] **`return`** [*expr*] **`exit`** [*n*]

Types



Variable Declarations

Implicit	*id*=...	Create at assignment
Explicit	**declare** [*attr*] *id*=... **typeset** [*attr*] *id*=...	**-a**: array **-i**: integer **-r**: readonly **-x**: export
	export *id*[=...]	Set environment for later commands
	readonly [*attr*] *id*=...	**-a**: array
	local [*attr*] *id*=...	Same options as **declare**
Remove	**unset** *id*	Also for arrays
	declare -n *id*	Not for arrays

Type Conversions

Type	Value	String	Integer	Array
String	Integer	Identity	As number	One-element array
	Other single word		Error	
	Blank separated words		Error	Multi-element array
Integer		As string	Identity	Singleton array
Array		First element	First element	Identity

Writing Subroutines

- Declaration: [**`function`**] *id* **`() { …; }`**
  - Can contain explicit **`return`**
  - Can declare variables with **`local`**
- Arguments: not declared
  - Positional parameters in **`$1`**, **`$2`**, …
  - At top level, **`$1`**, **`$2`**, … = command line arguments
- Listing functions:
  - **`declare -f`** / **`typeset -f`**
  - List names only: **`declare -F`** / **`typeset -F`**
- Exporting functions to environment: **`export`**

Operators

- Arithmetic operators (in **((…))**):
  - C operators, including **++**, **!**, **~**, **+**, **-**, *****, **/**, **<<**, **>>**, **<=**, **<**, etc., **&**, **|**, **&&**, **||**, **?:**, **=**, ***=**, etc.
- Conditional operators (in **[[…]]**):
  - File test operators: e.g., **-e** exists, **-d** is directory
  - Unary string operators: e.g., **-z** is zero
  - String comparison: **==**, **!=**, **<**, **>**
  - Arithmetic comparison: **-eq**, **-ne**, **-lt**, **-le**, **-gt**, **-ge**
- Control operators (in pipeline):
  - **||**, **&**, **&&**, **;**, **;;**, **(**, **)**, **|**, *newline*

Input and Output

- Output: **`echo "hello, world!"`**
  - Supress newline: **`echo -n "hi"`**
- Input from stdin to *id*: **`read`** *id*
  - Input from another file: **`read -u`** *fd id*
- Menu/read loop:

```
#!/bin/bash
select myInput in apple banana; do
  echo "You selected $myInput"
done
```

```
1) apple
2) banana
#? 2
You selected banana
#? 1
You selected apple
#? ^D
```

String Manipulation

- The bash manual talks about all variables as "parameters", not just positional parameters
  - Besides simple interpolation (**$***id* or **${***id***}**), bash also manipulates strings during parameter expansion
  - Expand at top level or in **"…"**, but not in **'…'**
- Return length: **${#***id***}**
- Extract substring: **${***id***:***offset***:***length***}**
- Remove matching prefix or postfix:
  - Shortest prefix: **${***id***#***pat***}**
  - Longest prefix: **${***id***##***pat***}**
  - Shortest suffix: **${***id***%***pat***}**
  - Longest suffix: **${***id***%%***pat***}**

Arrays

- Initialization:
  - From zero: *id*=(*expr* ... *expr*)
  - Indexed: *id*=([*index*]=*expr* ... [*index*]=*expr*)
- Indexing:
  - Zero-based, non-contiguous, integers only
  - Write: *id*[*index*]=*expr*
  - Read: $*id*[*index*]
- Deleting:
  - Element: **unset** *id*[*index*]
  - Entire array: **unset** *id*

Reference

Bash Documentation

- Manual page: **`man bash`**
- http://www.gnu.org/software/bash/manual/bashref.html

Evaluating Bash

Strengths

- Good at manipulating files and processes
- Same language for interaction and script

Weaknesses

- Lack of data structures
- Difficult to manipulate strings
- No debugger
- No static checker

Last Slide

- No announcements.
- Today's lecture
  - Bash
- Next lecture
  - Review (requests?)