UNIX® Shell and Utilities

Introduction to the UNIX Shell Command Language and Utilities, as Specified by POSIX.1 XCU

Introduction: What is a Shell?

- Command interpreter on UNIX and UNIX-like systems
- Allows users to control their systems
- Both a UI and an API for UNIX

Introduction: Brief History of UNIX Shells

- Concept from Multics
- Thomson shell (sh), 1971
- Bourne shell (sh), 1977
- C shell (csh), 1978
- Korn shell (ksh), 1983
- Almquist shell (ash), 1989
- GNU Bourne-Again SHell (bash), 1989

Introduction:
Shell Input Methods

- Command file
- Command string
- Standard input
- Interactive input

Basic Concepts: File Descriptors

- Integers that identify open files and details thereof
- Every UNIX process, including the shell and utilities, has a set of file descriptors
- Each shell command has its own set of file descriptors
- Three standard file descriptors are provided by the system:

fd 0	Standard input
fd 1	Standard output
fd 2	Standard error

Basic Concepts: Exit Status

- Every UNIX process exits with a numeric “exit status” between 0 and 255
- Each shell command has its own exit status
- Standard exit statuses:

0	Success
1–125	Failure
126	Command found but not executable
127	Command not found
128–255	Command terminated by signal

Basic Concepts: Shell Execution Environment

- The current “state” of the shell
- Consists of:
  - Open files
  - Working directory
  - Shell variables
  - Shell functions
  - …

Basic Concepts: Shell Execution Environment

- Subshell environment:
  - Created as a duplicate of the shell environment
  - Changes made to the subshell environment do not affect the shell environment

String Matching: Pattern Matching Notation

- A string of ordinary characters matches itself
- A <question-mark> is a pattern that matches any character
- An <asterisk> is a pattern that matches any string, including the null string
- A bracket expression matches any character

Basic Shell Grammar: Comments

- Text after an unquoted '#' character
- Ignored by shell

Basic Shell Grammar: Shell Commands

- Everything in *shell command* language is centered around *shell commands*
- Simple commands
  - Most simply:

    command-name [*argument1*] ...
  - *command-name* is executed with arguments, if any

Basic Shell Grammar: Quoting

- Removes the special meaning of certain characters or words
- Escape character (backslash)
  - Preserves the literal meaning of the following character
  - If a newline follows the backslash, the shell interprets it as a line continuation
- Single quotes
  - Preserve the literal meaning of each character within the quotes
  - A single quote cannot appear within single quotes

Basic Shell Grammar: Quoting

- Double quotes
  - Preserve the literal meaning of each character within the quotes, with exceptions
  - The dollar sign ('$') and backquote ('`') characters retain their special meaning introducing word expansions
  - The backslash ('\') character retains its special meaning as an escape character

Parameters: Assignment

- Actually part of simple commands, e.g.:

```
foo=1
foo=1 bar=2
baz=3 qux=4 cmdname arg1 arg2
```

- If there is a command name, variables are set only in the environment of the command
- If there is no command name, variable assignments affect the current execution environment

Parameters: Expansion

- Simple parameter

  `${parameter}`

- Use default values

  `${parameter:-word}`

- String length

  `${#parameter}`

Parameters: Expansion

- Remove smallest prefix pattern

  `${parameter#word}`
- Remove largest prefix pattern

  `${parameter##word}`
- Remove smallest suffix pattern

  `${parameter%word}`
- Remove largest suffix pattern

  `${parameter%%word}`

Parameters: Types

- Positional parameters
  - Decimal values greater than 0
  - Initially assigned when shell is invoked
  - Temporarily replaced when a shell function is invoked
  - Can be reassigned with the `set` utility

Parameters: Types

- Special parameters

  `@` All positional parameters
  `*` All positional parameters
  `#` The number of positional parameters
  `?` The exit status of the most recent pipeline
  `0` The name of the shell or shell script
  …

Parameters: Types

- Variables
  - Set by the user or application with parameter assignment
- Environment variables

`IFS`	List of characters used for field splitting
`LANG`	Default value for internationalization variables
`LINENO`	Current line number within a script
`PATH`	String affecting command search
`PS1`	Initial interactive shell prompt
`PS2`	Subsequent interactive shell prompt
`PWD`	Current working directory
…	

Word Expansions: Command Substitution

- Allows the standard output of a command, less any trailing <newline> characters, to be substituted for the command name itself

  `$(`*`command`*`)`
  `` `command` ``

- *command* is executed in a subshell environment
- Only field splitting and pathname expansion are performed on the results, and only if the substitution does not occur inside double-quotes

Word Expansions: Arithmetic Expansion

- Evaluates an arithmetic expression and substitutes its value

  `$((`*`expression`*`))`

- Performs long integer arithmetic with shell variables of the forms “x” and “$x” and decimal (e.g. “42”), octal (e.g. “052”), and hexadecimal (e.g. “0x2A”) constants

Word Expansions: Field Splitting

- Characters in the *IFS* are used as delimiters to split the results of previous expansions into fields
  1. If the value of *IFS* is null, no field splitting is performed
  2. If *IFS* is unset or its value is <space>, <tab>, and <newline>, any sequence of these whitespace characters at the beginning or end of the input is ignored and any such sequence within the input delimits a field

     For example, the input:

     ```
     <tab>foo<space><space>bar<space><newline>
     ```

     yields two fields: “**foo**” and “**bar**”

Word Expansions: Field Splitting

- Characters in the *IFS* are used as delimiters to split the results of previous expansions into fields

  3. Otherwise:

     - Whitespace (<space>, <tab>, or <newline>) *IFS* characters found at the beginning or end of the input are ignored
     - Non-whitespace *IFS* characters found in the input, along with any adjacent whitespace *IFS* characters, delimit fields
     - Whitespace *IFS* characters found in the input delimit fields

  For example, if the value of *IFS* is "`:<space>`", the input:

  `foo:bar<space>:<space>baz:<space>:qux<space>quux`

  yields six fields: "**foo**", "**bar**", "**baz**", "", "**qux**", and "**quux**"

Word Expansions: Pathname Expansion

- Each field in the input is expanded using pattern matching notation to match files, except:
  - A <slash> character in a pathname is not matched by <question-mark> or <asterisk> special characters or bracket expressions
  - A leading <period> in a filename is not matched by <question-mark> or <asterisk> special characters or certain bracket expressions
  - If the pattern string does not match any pathnames, it is left unchanged

Redirection

- Redirecting input

  `[`*n*`]<`*word*

- Redirecting output

  `[`*n*`]>`*word*

  `[`*n*`]>|`*word*

- Appending redirected output

  `[`*n*`]>>`*word*

Redirection

- Here-Document
  - Redirection of lines in a shell input file

    ```
    [n]<<word
    here-document
    delimiter
    ```

    ```
    [n]<<-word
    here-document
    delimiter
    ```

  - If no characters in *word* are quoted, parameter expansion, command substitution, and arithmetic substitution are performed on *here-document*

Shell Commands: Pipelines

- A sequence of one or more commands separated by the '|' operator

  [!] *command1* **[** | *command2* **...]**

- Exit status:
  - If the reserved word **!** does not precede the pipeline, the exit status is the exit status of the last command
  - Otherwise, the exit status is the logical NOT of the exit status of the last command

Shell Commands: Lists

- AND list
  - A sequence of one or more pipelines separated by the “&&” operator

    command1 [&& *command2*] ...
  - Exit status: that of the last command
- OR list
  - A sequence of one or more pipelines separated by the “||” operator

    command1 [|| *command2*] ...
  - Exit status: that of the last command

Shell Commands: Lists

- Sequential list
  - A sequence of one or more AND-OR lists separated by the '`;`' operator and optionally terminated by '`;`' or <newline>

    command1 [; *command2*] ...
  - Exit status: that of the last command
- Asynchronous list
  - A sequence of one or more AND-OR lists separated by the '&' operator and optionally terminated by '&' or <newline>

    command1 [& *command2*] ...
  - Exit status: zero

Shell Commands: Compound Commands

- Grouping commands
  - Execution in a subshell environment:

    `(`*`compound-list`*`)`
  - Execution in the current process environment:

    `{`*`compound-list`*`;}`
  - Exit status: that of *compound-list*

Shell Commands: Compound Commands

- **if** conditional construct

```
if compound-list
then
    compound-list
[elif compound-list
then
    compound-list] ...
[else
    compound-list]
fi
```

Shell Commands: Compound Commands

- **case** conditional construct

```
case word in
    [(]pattern1) compound-list;;
    [[(]pattern[ | pattern] ... )
        compound-list;;] ...
    [[(]pattern[ | pattern] ... )
        compound-list]
esac
```

Shell Commands: Compound Commands

- **while** loop

```
while compound-list-1
do
    compound-list-2
done
```

- **until** loop

```
until compound-list-1
do
    compound-list-2
done
```

Shell Commands: Compound Commands

- **for** loop

```
for name [ in [word ... ]]
do
    compound-list
done
```

Shell Commands: Function Definition Command

- A function is a user-defined command to call a compound command with new positional parameters
- A function is defined with a “function definition command”

```
fname() compound-command[io-redirect ...]
```

- *compound-command* is commonly a grouping command that executes in the current process environment, e.g.:

```
fname()
{
    compound-list
}
```

Command Search and Execution: General Procedure

- If the command name does not contain any <slash> characters, the shell tries to execute a special built-in utility, a shell function, or an external utility found in a directory listed in the *PATH*
- If the command name contains at least one <slash>, the shell executes the external utility

Command Search and Execution: External Utilities

- Separate utility environment
- Current user must have execute permission on the file
- Many OS kernels read a “magic number” from the first few bytes of an executable file to determine its type
  - Linux does this using filesystem “binfmt” modules
  - If the magic number is the ASCII string “#!” and the first line is of the following form:

    ```
    #! interpreter [argument]
    ```

    interpreter is called with *argument*, if any, the command name, and the remaining command arguments, if any

Command Search and Execution: External Utilities

- If the utility cannot be executed and is a text file, a new shell is executed with the command name as its first operand

UNIX® Shell and Utilities: Introduction to the UNIX Shell Command Language and Utilities, as Specified by POSIX.1 XCU