Fix Me! work in progress…

Parsing and execution

Nearly everything in Bash grammar can be broken down to a "simple command". The only thing Bash has to expand, evaluate and execute is the simple command.

Simple command expansion

- http://lists.gnu.org/archive/html/bug-bash/2013-01/msg00040.html (http://lists.gnu.org/archive/html/bug-bash/2013-01/msg00040.html)
- http://lists.research.att.com/pipermail/ast-developers/2013q2/002456.html (http://lists.research.att.com/pipermail/ast-developers/2013q2/002456.html)

This step happens after the initial command line splitting.

The expansion of a simple command is done in four steps (interpreting the simple command **from left to right**):

1. The words the parser has marked as **variable assignments** and **redirections** are saved for later processing.
   - variable assignments precede the command name and have the form `WORD=WORD`
   - redirections can appear anywhere in the simple command
2. The rest of the words are expanded. If any words remain after expansion, the first word is taken to be the **name of the command** and the remaining words are the **arguments**.
3. Redirections are performed.
4. The text after the `=` in each variable assignment undergoes tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal before being assigned to the variable.

If **no command name** results after expansion:

- The variable assignments affect the **current shell** environment.
  - This is what happens when you enter only a variable assignment at the command prompt.
  - Assignment to readonly variables causes an error and the command exits non-zero.
- Redirections are performed, but do not affect the current shell environment.

    - that means, a `> FILE` without any command **will** be performed: the `FILE` will be created!
- The command exits
    - with an exit code indicating the redirection error, if any
    - with the exit code of the last command-substitution parsed, if any
    - with exit code 0 (zero) if no redirection error happened and no command substitution was done

Otherwise, if a command name results:

- The variables saved and parsed are added to the environment of the executed command (and thus do not affect the current environment)
    - Assignment to readonly variables causes an error and the command exits with a non-zero error code.
    - **Assignment errors** in non-POSIX modes cause the *enclosing commands (e.g. loops) to completely terminate*
    - **Assignment errors** in (non-interactive) POSIX mode cause *the entire script to terminate*

The behavior regarding the variable assignment errors can be tested:

http://lists.gnu.org/archive/html/bug-bash/2013-01/msg00054.html (http://lists.gnu.org/archive/html/bug-bash/2013-01/msg00054.html)

This one exits the script completely

```
#!/bin/sh
# This shell runs in POSIX mode!

echo PRE

# The following is an assignment error, since there is no digit '9'
# for a base eight number!
foo=$((8#9))

echo POST
```

This one terminates only the enclosing compound command (the `{ …; }`):

```
#!/bin/bash
# This shell runs in native Bash-mode!

echo PRE

# The following is an assignment error!
# The "echo TEST" won't be executed, since the { ...; } is terminated
{ foo=$((8#9)); echo TEST; }

echo POST
```

Simple command execution

If a parsed simple command contains no slashes, the shell attempts to locate and execute it:

- shell functions
- shell builtin commands
- check own hash table
- search along `PATH`

As of Bash Version 4, when a command search fails, the shell executes a shell function named `command_not_found_handle()` using the failed command as arguments. This can be used to provide user friendly messages or install software packages etc. Since this function runs in a separate execution environment, you can't really influence the main shell with it (changing directory, setting variables).

Fix Me! to be continued

See also

- Internal: Redirection
- Internal: Introduction to expansions and substitutions

Discussion

Rodolfo (http://www.facebook.com/profile.php?id=100003418804641), 2013/03/21 08:39 ()

What a joy to find such clear thkinnig. Thanks for posting!

Oleg Dunauskas, 2015/02/08 16:43 ()

You wrote: → The text after the = in each variable assignment undergoes tilde expansion, parameter expansion, command substitution, arithmetic expansion, and quote removal before being assigned to the variable. What is about path expansion?

Jan Schampera, 2015/02/08 17:37 ()

Hi,

there's no pathname expansion on assignment. Example:

```
> foo=*
> echo "$foo" # note the quotes, otherwise the '*' would be ex
panded here (but still, not above on assignment)
*
```

Oleg Dunauskas, 2015/02/09 15:24 ()

Thanks for explanation.

syntax/grammar/parser_exec.txt Last modified: 2019/10/31 16:18 by ersen

This site is supported by Performing Databases - your experts for database administration

Bash Hackers Wiki

Except where otherwise noted, content on this wiki is licensed under the following license:
GNU Free Documentation License 1.3