Bash's behaviour

Fix Me! incomplete

Bash startup modes

Login shell

As a "login shell", Bash reads and sets (executes) the user's profile from `/etc/profile` and one of `~/.bash_profile` , `~/.bash_login` , or `~/.profile` (in that order, using the first one that's readable!).

When a login shell exits, Bash reads and executes commands from the file `~/.bash_logout` , if it exists.

Why an extra login shell mode? There are many actions and variable sets that only make sense for the initial user login. That's why all UNIX® shells have (should have) a "login" mode.

Methods to start Bash as a login shell:

- the first character of `argv[0]` is `-` (a hyphen): traditional UNIX® shells start from the `login` binary
- Bash is started with the `-l` option
- Bash is started with the `--login` option

Methods to test for login shell mode:

- the shell option `login_shell` is set

Related switches:

- `--noprofile` disables reading of all profile files

Interactive shell

When Bash starts as an interactive non-login shell, it reads and executes commands from `~/.bashrc` . This file should contain, for example, aliases, since they need to be defined in every shell as they're not inherited from the parent shell.

The feature to have a system-wide `/etc/bash.bashrc` or a similar system-wide rc-file is specific to vendors and distributors that ship *their own, patched variant of Bash*. The classic way to have a system-wide rc file is to `source /etc/bashrc` from every user's `~/.bashrc` .

Methods to test for interactive-shell mode:

- the special parameter `$-` contains the letter `i` (lowercase I)

Related switches:

- `-i` forces the interactive mode
- `--norc` disables reading of the startup files (e.g. `/etc/bash.bashrc` if supported) and `~/.bashrc`
- `--rcfile` defines another startup file (instead of `/etc/bash.bashrc` and `~/.bashrc`)

SH mode

When Bash starts in SH compatiblity mode, it tries to mimic the startup behaviour of historical versions of `sh` as closely as possible, while conforming to the POSIX® standard as well. The profile files read are `/etc/profile` and `~/.profile`, if it's a login shell.

If it's not a login shell, the environment variable ENV is evaluated and the resulting filename is used as the name of the startup file.

After the startup files are read, Bash enters the POSIX(r) compatiblity mode (for running, not for starting!).

Bash starts in `sh` compatiblity mode when:

- the base filename in `argv[0]` is `sh` (❗ NB: `/bin/sh` may be linked to `/bin/bash`, but that doesn't mean it acts like `/bin/bash` ❗)

POSIX mode

When Bash is started in POSIX® mode, it follows the POSIX® standard for startup files. In this mode, **interactive shells** expand the ENV variable and commands are read and executed from the file whose name is the expanded value.
No other startup files are read. Hence, a non-interactive shell doesn't read any startup files in POSIX® mode.

Bash starts in POSIX® mode when:

- the commandline option `--posix` is specified
- the environment variable POSIXLY_CORRECT is set

Quick startup file reference

- Eventual system-wide rc-files are usually read when `~/.bashrc` would be read (at least Debian GNU/Linux behaves like that)
- Regardless of the system-wide files in `/etc` which are always read, Bash usually reads the first file found, when multiple choices are given (for user files in `~/`)

Mode	`/etc/profile`	`~/.bash_profile`	`~/.bash_login`	`~/.profile`	`~`
Login shell	X	X	X	X	

Mode	`/etc/profile`	`~/.bash_profile`	`~/.bash_login`	`~/.profile`	`~/.`
Interactive shell	-	-	-	-	
SH compatible login	X	-	-	X	
SH compatible	-	-	-	-	
POSIX® compatiblity	-	-	-	-	

Bash run modes

Normal Bash

POSIX run mode

In POSIX® mode, Bash follows the POSIX® standard regarding behaviour and parsing (excerpt from a Bash maintainer's document):

```
Starting Bash with the `--posix' command-line option or executing `se
t
-o posix' while Bash is running will cause Bash to conform more close
ly
to the POSIX standard by changing the behavior to match that specifie
d
by POSIX in areas where the Bash default differs.

When invoked as `sh', Bash enters POSIX mode after reading the startu
p
files.

The following lists what's changed when Bash is in `POSIX mode':

  1. When a command in the hash table no longer exists, Bash will
     re-search `$PATH' to find the new location.  This is also
     available with `shopt -s checkhash'.

  2. The message printed by the job control code and builtins when a
job
     exits with a non-zero status is `Done(status)'.

  3. The message printed by the job control code and builtins when a
job
     is stopped is `Stopped(SIGNAME)', where SIGNAME is, for example,
     `SIGTSTP'.

  4. The `bg' builtin uses the required format to describe each job
     placed in the background, which does not include an indication o
f
     whether the job is the current or previous job.

  5. Reserved words appearing in a context where reserved words are
     recognized do not undergo alias expansion.

  6. The POSIX `PS1' and `PS2' expansions of `!' to the history numbe
r
     and `!!' to `!' are enabled, and parameter expansion is performe
d
     on the values of `PS1' and `PS2' regardless of the setting of th
e
     `promptvars' option.

  7. The POSIX startup files are executed (`$ENV') rather than the
     normal Bash files.

  8. Tilde expansion is only performed on assignments preceding a
     command name, rather than on all assignment statements on the li
ne.

  9. The default history file is `~/.sh_history' (this is the default
     value of `$HISTFILE').

 10. The output of `kill -l' prints all the signal names on a single
     line, separated by spaces, without the `SIG' prefix.
 11. The `kill' builtin does not accept signal names with a `SIG'
     prefix.

 12. Non-interactive shells exit if FILENAME in `.' FILENAME is not
     found.

 13. Non-interactive shells exit if a syntax error in an arithmetic
     expansion results in an invalid expression.

 14. Redirection operators do not perform filename expansion on the w
ord
     in the redirection unless the shell is interactive.

 15. Redirection operators do not perform word splitting on the word
in
     the redirection.

 16. Function names must be valid shell names.  That is, they may not
     contain characters other than letters, digits, and underscores,
and
     may not start with a digit.  Declaring a function with an invali
d
     name causes a fatal syntax error in non-interactive shells.

 17. POSIX special builtins are found before shell functions during
     command lookup.

 18. If a POSIX special builtin returns an error status, a
     non-interactive shell exits.  The fatal errors are those listed
in
     the POSIX standard, and include things like passing incorrect
     options, redirection errors, variable assignment errors for
     assignments preceding the command name, etc.

 19. If `CDPATH' is set, the `cd' builtin will not implicitly append
     the current directory to it.  This means that `cd' will fail if
no
     valid directory name can be constructed from any of the entries
in
     `$CDPATH', even if the a directory with the same name as the nam
e
     given as an argument to `cd' exists in the current directory.

 20. A non-interactive shell exits with an error status if a variable
     assignment error occurs when no command name follows the assignm
ent
     statements.  A variable assignment error occurs, for example, wh
en
     trying to assign a value to a readonly variable.

 21. A non-interactive shell exits with an error status if the iterat
ion
     variable in a `for' statement or the selection variable in a
     `select' statement is a readonly variable.
 22. Process substitution is not available.

 23. Assignment statements preceding POSIX special builtins persist i
n
     the shell environment after the builtin completes.

 24. Assignment statements preceding shell function calls persist in
the
     shell environment after the function returns, as if a POSIX
     special builtin command had been executed.

 25. The `export' and `readonly' builtin commands display their outpu
t
     in the format required by POSIX.

 26. The `trap' builtin displays signal names without the leading `SI
G'.

 27. The `trap' builtin doesn't check the first argument for a possib
le
     signal specification and revert the signal handling to the origi
nal
     disposition if it is, unless that argument consists solely of
     digits and is a valid signal number.  If users want to reset the
     handler for a given signal to the original disposition, they
     should use `-' as the first argument.

 28. The `.' and `source' builtins do not search the current director
y
     for the filename argument if it is not found by searching `PAT
H'.

 29. Subshells spawned to execute command substitutions inherit the
     value of the `-e' option from the parent shell.  When not in POS
IX
     mode, Bash clears the `-e' option in such subshells.

 30. Alias expansion is always enabled, even in non-interactive shell
s.

 31. When the `alias' builtin displays alias definitions, it does not
     display them with a leading `alias ' unless the `-p' option is
     supplied.

 32. When the `set' builtin is invoked without options, it does not
     display shell function names and definitions.

 33. When the `set' builtin is invoked without options, it displays
     variable values without quotes, unless they contain shell
     metacharacters, even if the result contains nonprinting characte
rs.

 34. When the `cd' builtin is invoked in LOGICAL mode, and the pathna
me
     constructed from `$PWD' and the directory name supplied as an
     argument does not refer to an existing directory, `cd' will fail
     instead of falling back to PHYSICAL mode.

 35. When the `pwd' builtin is supplied the `-P' option, it resets
     `$PWD' to a pathname containing no symlinks.

 36. The `pwd' builtin verifies that the value it prints is the same
as
     the current directory, even if it is not asked to check the file
     system with the `-P' option.

 37. When listing the history, the `fc' builtin does not include an
     indication of whether or not a history entry has been modified.

 38. The default editor used by `fc' is `ed'.

 39. The `type' and `command' builtins will not report a non-executab
le
     file as having been found, though the shell will attempt to
     execute such a file if it is the only so-named file found in
     `$PATH'.

 40. The `vi' editing mode will invoke the `vi' editor directly when
     the `v' command is run, instead of checking `$FCEDIT' and
     `$EDITOR'.

 41. When the `xpg_echo' option is enabled, Bash does not attempt to
     interpret any arguments to `echo' as options.  Each argument is
     displayed, after escape characters are converted.


There is other POSIX behavior that Bash does not implement by default
even when in POSIX mode.  Specifically:

  1. The `fc' builtin checks `$EDITOR' as a program to edit history
     entries if `FCEDIT' is unset, rather than defaulting directly to
     `ed'.  `fc' uses `ed' if `EDITOR' is unset.

  2. As noted above, Bash requires the `xpg_echo' option to be enable
d
     for the `echo' builtin to be fully conformant.


Bash can be configured to be POSIX-conformant by default, by specifyi
ng
the `--enable-strict-posix-default' to `configure' when building.
```

Fix Me! help me to find out what breaks in POSIX® mode!

The POSIX® mode can be switched on by:

- Bash starting as `sh` (the basename of `argv[0]` is `sh`)
- starting Bash with the commandline option `--posix`
- on startup, the environment variable POSIXLY_CORRECT is set
- the command `set -o posix`

Tests for the POSIX® mode:

- the variable SHELLOPTS contains `posix` in its list

Restricted shell

In restricted mode, Bash sets up (and runs) a shell environment that's far more controlled and limited than the standard shell mode. It acts like normal Bash with the following restrictions:

- the `cd` command can't be used to change directories
- the variables SHELL, PATH, ENV and BASH_ENV can't be set or unset
- command names that contain a `/` (slash) can't be called (hence you're limited to `PATH`)
- filenames containing a `/` (slash) can't be specified as argument to the `source` or `.` builtin command
- filenames containing a `/` (slash) can't be specified as argument to the `-p` option of the `hash` builtin command
- function definitions are not inherited from the environment at shell startup
- the environment variable SHELLOPTS is ignored at startup
- redirecting output using the `>` , `>|` , `<>` , `>&` , `&>` , and `>>` redirection operators isn't allowed
- the `exec` builtin command can't replace the shell with another process
- adding or deleting builtin commands with the `-f` and `-d` options to the enable builtin command is forbidden
- using the `enable` builtin command to enable disabled shell builtins doesn't work
- the `-p` option to the `command` builtin command doesn't work
- turning off restricted mode with `set +r` or `set +o restricted` is (of course) forbidden

The "-r" restrictions are turned on **after** Bash has read its startup files.

When the command that is run is a shell script, then the restrictions are **turned off** for the (sub-)shell that runs that shell script.

The restricted shell can be switched on by:

- calling Bash as `rbash` (the basename of `argv[0]` is `rbash`)
- calling Bash with the `-r` option
- calling Bash with the `--restricted` option

Tests for restricted mode:

- the special parameter `$-` contains the letter `r` (lowercase R)
- the shell option `restricted_shell` is set and can be checked by the `shopt` builtin command

Discussion

Ed, 2012/08/23 09:30 ()

[quote]Methods to test for login shell mode:

```
  the shell option login_shell is set[/quote]
```

Well that tells mw WHAT is set, but not HOW to check for it.

Which is what I actually came here to find out!

Jan Schampera, 2012/08/25 17:18 ()

```
if shopt -q login_shell; then
...
```

`shopt` sets an exit status of 0/1 according to on/off

Leonardo D'iaz, 2012/09/20 14:51 ()

Debian's /etc/profile also source /etc/bash.bashrc and /etc/profile.d/*.sh

/etc/bash.bashrc does some PS1 magic, sets bash_completion(s) and command-not-found (suggest packages)

scripting/bashbehaviour.txt Last modified: 2015/08/04 10:03 by thebonsai