THE LINUX SHELLS

Use, Understand, Customize

Rayan Mac

2020-03-25

What is a shell ?

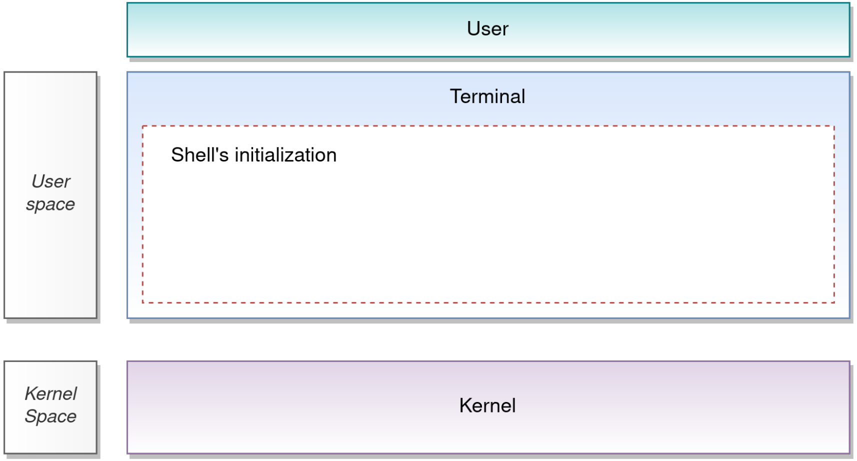
What is a shell ?

A command interpreter

PART 1 : Living like a shell

Outline

Initialization



Initialization - Getting environment from start-up files (Bash)

Login shells

- After an immediate login, prepended with a '-' character (echo $0)
- Read: System login scripts : /etc/profile
  User login scripts : ~/.bash_profile, ~/.bash_login, or ~/.profile (the first existing file)

Non login shells

- Started on demand, when user has already logged in
- Only read ~/.bashrc

Common practice:

```
    # ~/.bash_profile
    [[ -f ~/.bashrc ]] && . ~/.bashrc

```

Initialization - Great reminders

Man bash:

```
INVOCATION
A login shell is one whose first character of argument zero is a −, or one started with the −−login option.
[...]
When  bash  is  invoked  as an interactive login shell, or as a non−interactive shell with the −−login option, it first
  reads and executes commands
from the file /etc/profile, if that file exists.  After reading that file, it looks for ~/.bash_profile, ~/.bash_login, and
  ~/.profile, in that or
der,  and reads and executes commands from the first one that exists and is readable.  The −−noprofile option may
  be used when the shell is started
to inhibit this behavior.

```

Initialization - Great reminders

Source code:

```
if (login_shell < 0 && posixly_correct == 0)
{
  /* We do not execute .bashrc for login shells. */
  no_rc++;

  /* Execute /etc/profile and one of the personal login shell
     initialization files. */
  if (no_profile == 0)
  {
      maybe_execute_file (SYS_PROFILE, 1);

      if (act_like_sh)/* sh */
          maybe_execute_file ("~/.profile", 1);
      else if ((maybe_execute_file ("~/.bash_profile", 1) == 0) &&
               (maybe_execute_file ("~/.bash_login", 1) == 0))/* bash */
          maybe_execute_file ("~/.profile", 1);
  }
  sourced_login = 1;
}
```

Initialization - The shell execution environment

The shell has an execution environment, which consists of the following:

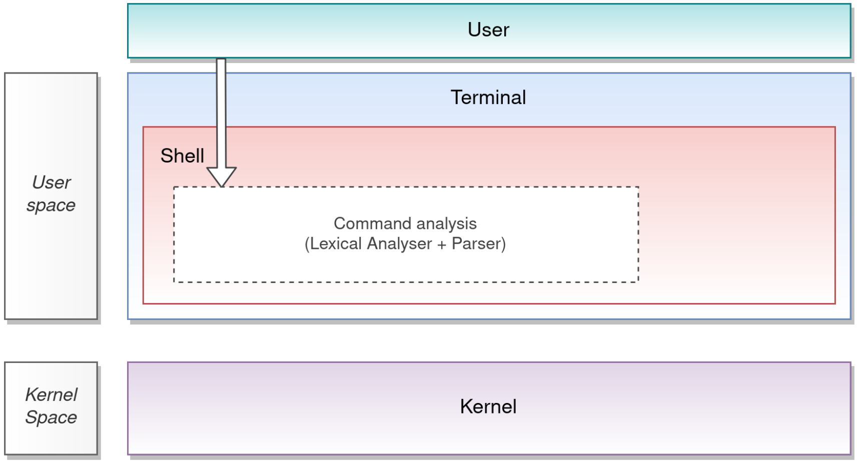
- OPENED FILES inherited by the shell at invocation
- CURRENT WORKING DIRECTORY as set by cd
- FILE CREATION MODE MASK, as set by umask
- CURRENT TRAPS, set by trap
- ENVIRONMENT AND SHELL VARIABLES
- SHELL FUNCTIONS AND ALIASES
- VARIOUS PROCESS IDs, those of background jobs, $$ and $PPID, etc...
- OTHER OPTIONS enabled by 'set' or 'shopt' builtins

Some useful set options

set -e set -o errexit	Exit immediately if a simple command exits with a non-zero status
set -n set -o noexec	Read commands but do not execute them; this can be used to check a script for syntax errors.
set -u set -o nounset	Treat unset variables as an error when performing parameter expansion.
set -v set -o verbose	Print shell input lines as they are read.
set -x set -o xtrace	Print a trace of simple commands and their arguments after they are expanded and before they are executed.

Common practice in writing and debugging: set -eu

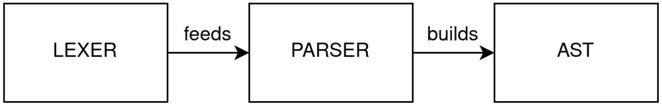
Command Parsing



Command Parsing - Input Analysis

1. Token Recognition (words are split on whitespaces)
2. Substitution (parameter and command)
3. Field splitting (words are split on $IFS variable)
4. Globbing
5. Execution (commands and control sructures)

Going deeper: The similarities with compilers



Lexer (Lexical Analyser): Breaks the input string to a series of token through lexical analysis
Parser: Grammatical analysis of the tokens to build the AST
AST (Abstract Syntax Tree) : A tree-like data structure that holds tokens and operations in order of the execution

Quiz time ! What is the output of these commands ?

```
IF=if
$IF true; then echo "hello world"; fi

```

Quiz time ! What is the output of these commands ?

```
IF=if
$IF true; then echo "hello world"; fi

```

Output: Error, unexpected token

Quiz time ! What is the output of these commands ?

```
file="foo.txt"  # foo.txt not being empty
head -n 10 ${file} > {file}
cat ${file}

```

Quiz time ! What is the output of these commands ?

```
file="foo.txt"  # foo.txt not being empty
head -n 10 ${file} > {file}
cat ${file}

```

Output: Nothing

Quiz time ! What is the output of these commands ?

```
STR="Hello        :great:world"
echo $STR
IFS=':' ; echo $STR

```

Quiz time ! What is the output of these commands ?

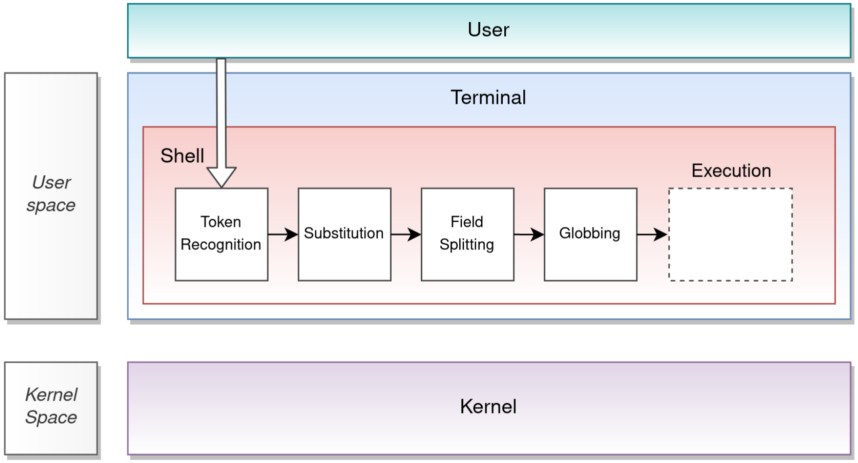
```bash
STR="Hello        :great:world"
echo $STR
IFS=':' ; echo $STR

```

Output:

```
Hello :great:world
Hello         great world
```

More challenges at https://mywiki.wooledge.org/BashPitfalls

Execution


User
Terminal
Shell
Execution
Token Recognition
Substitution
Field Splitting
Globbing
User space
Kernel Space
Kernel

Execution - External commands

fork / execve

Execution - Tracking with strace

```
$ strace bash
$ cat foo.txt > bar.txt
```

```
[..........]
open("bar.txt", O_WRONLY|O_CREAT|O_TRUNC, 0666) = 3
dup2(3, 1)                          = 1
close(3)                            = 0
execve("/bin/cat", ["cat", "foo.txt"], [/* 47 vars */]) = 0
[..........]
open("foo.txt", O_RDONLY)           = 3
[..........]
read(3, "Hello world\n", 131072)    = 12
write(1, "Hello world\n", 12)       = 12
read(3, "", 131072)                 = 0
munmap(0x7f5cd9dcf000, 139264)      = 0
close(3)                            = 0
close(1)                            = 0
close(2)                            = 0
exit_group(0)                       = ?
+++ exited with 0 +++
```

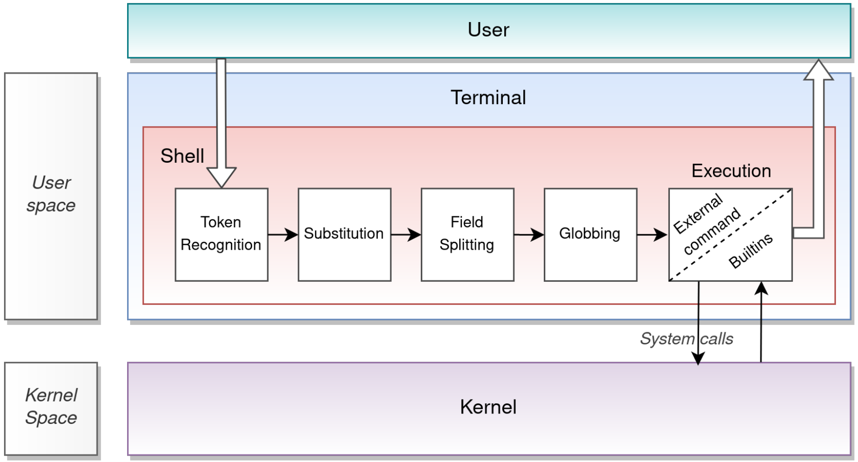
Execution - Built-ins

Executed directly in the shell itself

- Faster execution
- Can modify the shell's context

Check nature of a command: `$ type <cmd>`
Display list of shell built-ins: `$ help`

Overall process



PART 2 : A sea of shells

Outline

A brief of history

1971 : The Thompson shell

- First UNIX shell
- No scripting
- 900 lines of C source

1977 : Bourne Shell (sh)

- Command interpreter AND scripting language

+ Variables and command substitution
+ Control structures and loop

A brief of history

1978 : The C shell (csh) then Tenex C shell (tcsh)

- Scripting language "similar" to the C language
- Incompatible with sh

+ Command history in interactive use

1983: The Korn shell (ksh)

- Proprietary software until 2000 (then Common Public Licence)

+ Associative arrays

+ Floating point arithmetic

A brief of history

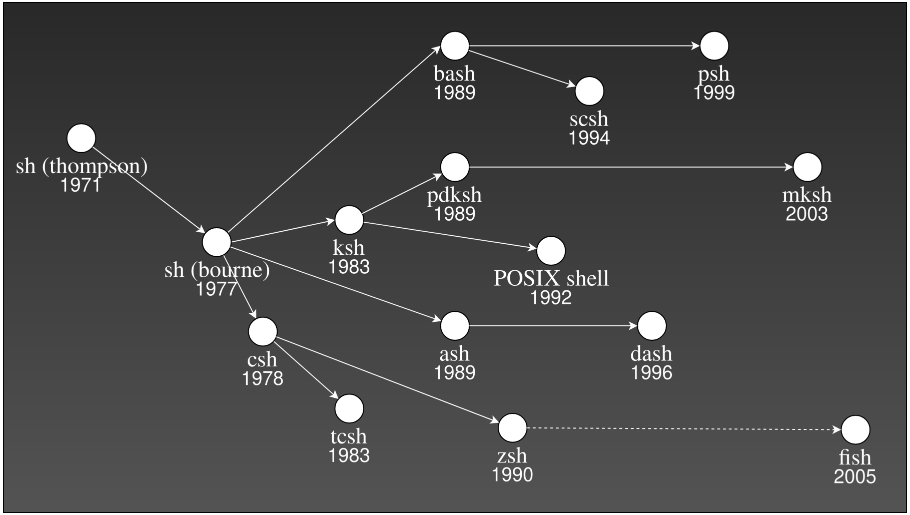
1989: The Almquist Shell (ash)

- A lightweight sh version
- Only implements POSIX features
- The Busybox's shell

1989: The Bourne Again Shell (bash)

- GNU GPLv3
- Enhanced version of sh
- Comes packaged as part of GNU

And many more !



What about compatibility ?

1992: Definition of what a POSIX shell shall be

When portability matters, avoid using a shell's specific feature
Try to execute your script using the bare /bin/sh (if existing)

Otherwise:

- POSIX shell standard available online
- Tools exist to assess POSIX compliance:
    - The ash/dash (and posh ?) shell
    - 'shellcheck' utility

What to expect from a shell today ?

Bash 5.0 Release, what's new ?

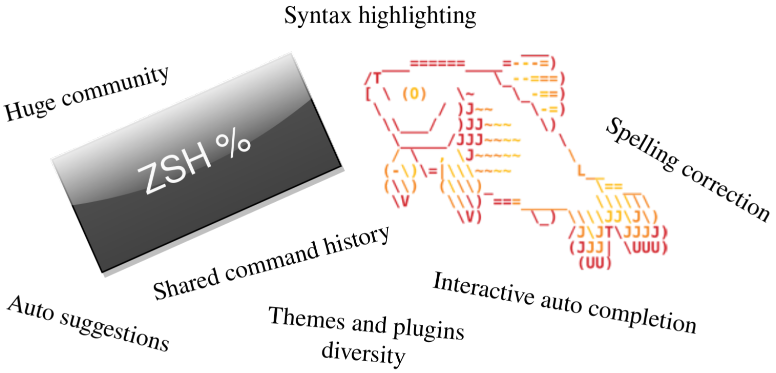
2019/01/07: v4.4 → v5.0 (stable)

- Bugfixes (potential out-of-bounds memory errors, ...)
- New shell variables: BASH_ARGV0, EPOCHSECONDS et EPOCHREALTIME
- Shell option 'globasciiranges' enabled by default (ensure [a-d] == [abcd])
- New options for the 'history' built-in

  ```
  history -d <start>-<end>
  ```
- `[...]`

More interactive features ? [Demo time]

Examples of the fish and zsh shells:



Which shell to use or test scripts with ?

Some suggestions:

- Embedded: ash (making local tests with dash if more convenient)
- Portability: sh when available (will lead you to a shell considered as POSIX)
- Daily use: (bash|ksh|fish|zsh|.*)

CONCLUSION

USE your shell efficiently, tweaking options when helpful

UNDERSTAND how it works to improve your scripting

CUSTOMIZE it to your needs and enjoy !

Thanks for your attention

Let's share !

References and useful resources

Books:

- Peter Seebach - "Beginning Portable Shell Scripting"
- Christophe Blaess - "Shells Linux et Unix"
- Arnold Robbins & Nelson H.F. Beebe - Classic Shell Scripting; // O'Reilly Edition

Links:

- https://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html
- https://developer.ibm.com/tutorials/l-linux-shells/#artrelatedtopics
- http://www.aosabook.org/en/bash.html
- https://github.com/Swoorup/mysh