CN/DCI/162

Shell Choice
A shell comparison

Arnaud Taddei

September 28, 1994

Abstract

This paper compares the interactive properties of the six best known shell-programs in the UNIX world. Clues are given to choose the best(s) one(s). The comparison is based on the description of each shell and some previous papers published in other HEP laboratories and in the internet newsgroup comp.unix.shell. As the shells are evolving this document will be upgraded periodically.

Contents

1 Shell choice, some clues

1.1 Candidates

There are six major candidates:

```
sh csh ksh bash tcsh zsh
```

There are other candidates like `rc`, the POSIX shell and the windowing shell (and probably a big 'zoo' of a-zsh). But I think they are less relevant in this context because `rc` is used in another UNIX-project, the windowing shell deals with widgets and we don't consider it here, the POSIX shell is not leading the subject. Moreover the shells we treat in this paper are seen as the most classical ones and are well-known.

They can be classified in different ways:

Family (according to syntax and grammar):

Bourne-shell flavour	**C-shell flavour**
`sh ksh bash zsh`	`csh tcsh`

Support and development:

Vendor support	**Public domain support**
`sh csh ksh`	`bash tcsh zsh`

Chronologically (how uptodate they are):

```
sh csh ksh tcsh bash zsh
```

1.2 Criteria for selection: the shell features

1 – Configurability
1.1 – Variables
1.2 – Aliases
1.3 – Functions
1.4 – Options
1.5 – User and global startup files
2 – Execution of commands
3 – Completion
4 – Line (and multiline) editing
5 – Name substitution or globbing
6 – History mechanism
7 – Redirections and pipes
8 – Spelling correction

9 – Prompt setings
10 – Job control
11 – Execution control
12 – Signal handling
13 – Miscellaneous

2 A brief description of all candidates

2.1 `sh`

The 'grand-daddy' of all modern shells. Father of the Bourne-shell flavour. Written by S. R. Bourne. It has a powerful built in language. sh is provided by vendors. It supplies the following features:

- Startup files (`.profile /etc/profile`)
- Command execution
- Monoline editing but very poor
- Filename, variable and conditional substitution
- Input/output redirection and flow control
- Signal handling
- Built-in commands

2.2 `csh`

The father of the C-shell flavour. It was born at UCB, written by Bill Joy, and the builtin language is close to the C syntax and grammar. csh is provided by vendors. It supplies the following features:

- Startup files
- Command aliasing
- Command execution
- Filename completion
- Monoline editing but very poor
- Command, filename and variable substitution or globbing
- History mechanism and substitution
- Input/output redirection and flow control
- Job control
- Built-in commands

Unfortunately, versions of some vendors are buggy and some vendors don't use the same startup files under `/etc` if any!

2.3 `ksh`

David Korn from AT&T improved `sh` (see 2.1). You may sse it as a Bourne and C-shell superset because it uses the Bourne syntax and add good interactive features from `csh`[1]. The POSIX shell is very much like `ksh`.

[1] But see later: 2.6.

ksh is also provided by vendors. Among the features, one can find:

- Startup files
- Aliasing, functions and options
- Command execution
- Filename completion
- Mono line editing
- Command, filename and variable substitution
- History mechanism
- Input/output redirection and flow control
- Job control
- Signal handling
- Built-in commands

2.4 bash

bash belongs to the Bourne shell flavour and means Bourne Again SHell. It was developped by the GNU project. It inherits Bourne shell features and syntax (see 2.1) and some features from both csh and tcsh (see 2.2 and 2.5). Like ksh it tries to be a POSIX compliant.

- specific startup files.
- startup files are the same for any platform.
- specific shell variables.
- some builtin commands like set and help.

Apart from that it behaves like ksh (see 2.3).

2.5 tcsh

In order to correct csh bugs and to improve its features, a lot of people decided to rewrite it. So tcsh was born because of its TENEX-style command line multiediting. It behaves like csh but adds the following features:

- specific startup files.
  /etc/tcshrc and /.tcshrc used to give interactive tcsh specific statements like setting tcsh-specific variables.
- startup files are the same for any platform.
- specific shell variables.
- specific builtin commands.
- enhanced completion mechanism
  The completion mechanism is programmable for commands, file names, variable names, user names, etc.
- multiline editing capabilities:
  Command line editing using emacs-style or vi-style key bindings.
- enhanced name substitution mechanism (globbing)
  The file expression syntax is enhanced: you can ask for names of files which are not containing a certain character and you can negate globbing patterns.
- enhanced history mechanism.

You can navigate (step up/down) through the command history list.

- spelling correction.
- enhanced prompt.
  Allows underlines, inverse video, bold, etc.

2.6 `zsh`

`zsh` is a very powerful shell. The syntax and grammar belongs to the Bourne shell family and so `zsh` is close to `sh`, `ksh`, `bash`. But is includes `csh` and `tcsh` features and it is very analogous to `tcsh`. The most important `zsh` features are:

- specific startup files.
- startup files are the same for any platform.
- specific shell variables.
- specific options.
- specific functions.
  these functions are auto-loaded when they are first referenced.
- specific builtins, e.g.
  `which -a cmd` gives all occurences of `cmd` in the path.
- enhanced completion mechanism
  Completion mechanism is programmable for commands, file names, variables names, user names, hostnames, key-bindings, options, user-specified keywords, etc.
  Menu completion. You can cycle through the possibilities while pressing the completion key (*TAB* like in tcsh see 2.5)
- multiline editing capabilities:
  Command line editing using emacs-style or vi-style key bindings.
- enhanced name substitution mechanism (globbing)
  File expression syntax is enhanced: you can request names of files which are not containing a certain character and you can negate globbing patterns.
  Recursive search can be done via `ls **/file`, etc.
- enhanced history mechanism.
  Searching through the visual history mechanism.
  Visual step up/down through the command history list.
  Command is brought up and not executed: you can re-edit it (see histverify option).
- enhanced redirections and piping.
- spelling correction.
- enhanced prompt.
  Allows underlines, inverse video, bold, etc.
  Can be put on the right of the screen!
- enhanced path hashing: incremental.

3 Comparison and conclusion

+++	very good
++	good
+	existing
-	weak
–	absent

Criteria	**Nb (see 1.2)**	**sh**	**ksh**	**bash**	**zsh**	**csh**	**tcsh**
Configurability	1	-	+	++	+++	+	++
Execution of commands	2	+	+	+	++	+	++
Completion	3	- -	+	++	+++	+	++
Line editing	4	-	+	++	++	-	++
Name substitution	5	+	+	++	++	+	++
History	6	- -	+	++	++	+	++
Redirections and pipes	7	+	+	+	++	+	+
Spelling correction	8	- -	- -	- -	+	- -	+
Prompt settings	9	+	+	+	++	+	++
Job control	10	- -	+	+	+	+	+
Execution control	11	+	+	+	+	+	+
Signal Handling	12	+	+	+	+	-	-

It is easily seen from the table that the dominant shells are tcsh and zsh.

The choice of a shell is not only determined by its features but also by the support yu can get.

Vendor shells are all slightly different (which is a pain when you try to homogeneise user environment).

Public domain shells are the same on all platforms. Thus although vendor shells may seem to be the easiest choice they are probably not the best one!

Comparison between `csh` and `tcsh`

`tcsh` is a public domain shell. The advantage is that you don't have to rely on vendors[2]. Another strong advantage is that it always reads files under `/etc` which is not the case for some `csh` shells provided by vendors like for example Silicon Graphics. Besides, `tcsh` provides a lot more features.

Comparison between `ksh` and `zsh`

Very often people ask the question, why `zsh` is better than `ksh`? Here are the three major arguments:

1 - `ksh` is a vendor supplied shell which doesn't exist on Sun platforms. There is a public domain `pdksh` but is it worth to support it as there are better public domain shells? `zsh` exists on all platforms and behaves the same way on all of them.

2 - `zsh` has many more features than `ksh` and for some features it is far better:

[2] This can also be a disadvantage as you have to provide the support. But it is worth the price to get `tcsh`.

- completion is better because you have to enter only the *TAB* key with `zsh` and with `ksh` you need to press *ESC* twice which can be boring when you are using this feature a lot of times. Moreover the completion mechanism with `zsh` is by far the most powerful as it can complete filenames in the context of a command (for example it will complete first a `.dvi` file if you are running a `dvips` command), usernames, command names, shell-options names, hostnames, command options names and you can even program it!
- line editing is better and not buggy with `zsh` because it has a multine editing capabilities (it has a builtin *ZLE* language) so one can edit a command line on more than one line. With `ksh` you need to set the line to be very long (which is somewhat ridiculous) and there are even some cases where it can break your session completely[3]!
- history mechanism is better in `zsh` as it inherited a lot of features from `csh`. Moreover it is easier to jump to commands in the history and to use history substitution which is not the case with `ksh`.

3 - `ksh` doesn't always read some startup files which means if you run a `rsh` command `ksh` doesn't read any file so it doesn't start your environment and you need to re-enter the PATH variable! This behaviour doesn't exist with `zsh` which means you are guaranted to have your environment set in any condition.

There are other nice features with `zsh` because it is very similar to `tcsh`. This means that C-shell fans can easily be converted to Bourne shell fans or the other way round using `zsh`

So `zsh` should be recommended to users rather than `ksh`.

`tcsh` versus `zsh`

Both are very close although if they belong to two different families. Another paper[4] is describing them in a more accurate way and shows their similarities. They are both excellent shells and are actually recommended for users at other HEP sites.

[3] For further information contact the author who can fully describe the bug
[4] Arnaud Taddei - Shell support, `tcsh` and `zsh` for pedestrians - 1994

A Summary of features

This table is rather exhaustive and is difficult to understand because a lot of features are not well-known but it was used to design the table in section 3 and helped to draw the conclusion[5].

Nb (see 1.2)	Features	sh	ksh	bash	zsh	csh	tcsh
1.1	Readonly variables	N	?	?	Y	N	Y
1.1	Specific shell variables	N	N	Y	Y	N	Y
1.1	With histverify, doesn't execute the line	N	?	Y	Y	N	Y
1.2	Aliases	N	Y	Y	Y	Y	Y
1.3	Auto-loaded functions	N	N	N	Y	N	N
1.3	Shell functions	Y(1)	Y	Y	Y	N	N
1.4	Options are available	N	Y	Y	Y	N(6)	N(6)
1.5	Input Files	N	Y(*)	Y	Y	Y(*)	Y
1.5	Specific startup files	N	Y	Y	Y	N	Y
1.5	Up to 9 startup files	N	N	N	Y	N	Y
1.5	Can avoid user startup files	N	N	Y	Y	N	N
1.5	Can specify startup files	N	Y	Y	N	N	N
1.5	Has non-interactive startup file	N	Y(5)	Y(5)	Y	Y	Y
1.5	Has non-login startup file	N	Y(5)	Y	Y	Y	Y
2	Automatic execution of a command when the current working directory is changed	N	N	N	Y	N	Y
2	Periodic command execution	N	N	N	Y	N	Y
2	Scheduled event list, which specifies commands which are to be executed at given times	N	N	N	Y	N	Y
2	Process Substitution	N	N	Y(2)	Y	N	N
2	Automatic process time reporting	N	?	Y	Y	N	N
2	Null command short-hands	N	N	N	Y	N	N
2	Incremental path hashing	N	?	Y	Y	N	?
2	Can cope with large argument lists	Y	Y	Y	Y	N	Y
3	Fully programmable Completion	N	N	N	Y	N	Y
3	User name look up	N	Y	Y	Y	Y	Y
3	Filename completion	N	Y	Y	Y	Y(1)	Y
3	Username completion	N	Y	Y	Y	Y(2)	Y
3	Hostname completion	N	Y	Y	Y	Y(2)	Y
3	History completion	N	N	Y	Y	N	Y
3	Command completion	N	N	Y	Y	N	Y
3	Variable completion	N	N	Y	Y	N	?
3	Option completion	N	N	N	Y	N	N
3	Interactive command, user name, file-name, hostname	N	Y	Y	Y	Y(2)	Y
3	Menu completion	N	N	N	Y	N	N

[5] For further information contact the author who will give more details in the next release of this documentation

Nb (see 1.2)	Features	`sh`	`ksh`	`bash`	`zsh`	`csh`	`tcsh`
4	Command line editing	N	Y(**)	Y	Y	N	Y
4	Command line editing with arrow keys, configurable	N	Y(**)	Y	Y	N	Y
4	Command line is multiline editing	N	N	Y	Y	N	Y
4	Vi Command line editing	N	Y	Y	Y	N	Y(3)
4	Emacs Command line editing	N	Y	Y	Y	N	Y
4	Rebindable Command line editing	N	N	Y	Y	N	Y
4	Sun Keyboard Hack	N	N	N	Y	N	Y
5	File/directory/user list in the middle of a typed command	N	Y	Y	Y	Y(2)	Y
5	Subset of ls	N	N	Y	Y	N	Y
5	Addition to the file expression syntax for a character not in a set of characters and the ability to negate a globbing pattern.	N	N	Y	Y	N	Y
5	Enhanced file inquiry operator	N	N	Y	Y	N	Y
5	Advanced globbing	N	N	Y	Y	N	Y
6	Command history	N	Y	Y	Y	Y	Y
6	Time stamps in the history list	N	N	N	N	Y	Y
6	Searching for the visual history mechanism	N	Y	Y	Y	N	Y
6	History completion	N	N	N	Y	N	Y
6	Incremental history search	N	N	Y	Y	N	Y
6	History is cached into memory	N	N	Y	Y	Y	Y
7	Command pipes and redirections	Y	Y	Y	Y	Y	Y
7	Generalized pipes	N	N	N	Y	N	N
7	"Sensible" Input/Output redirection	Y	Y	Y	Y	N	Y
8	Spelling Correction	N	N	N	Y	N	Y
9	Custom Prompt (easily)	N	Y	Y	Y	N	Y
9	Prompt on right size of screen	N	N	N	Y	N	N
10	Job control	N	Y	Y	Y	Y	Y
11	Execution Control	N	Y	Y	Y	N	Y
11	Good loops syntax shortcircuits	N	N	N	Y	N	N
12	Signal Handling	Y	Y	Y	Y	N	N
13	Check mailbox	N	Y	Y	Y	Y	Y
13	Underlying Syntax	B	B	B	B	C	C
13	Freely available	N	N(4)	Y	Y	N	Y
13	Native Language System support	N	N	Y	Y	N	Y

(*) Unfortunately versions of some vendors (e.g. DEC, SGI, SUN, HP) read some global files but not all if any.

(**) Unfortunately `ksh` is only a MONO line editing (even with bugs).

(1) This feature was not in the original version, but has since almost become standard.

(2) This feature is fairly new and is therefore not found on many versions of the shell.

(3) The vi emulation of this shell is thought by many to be incomplete.

(4) A version called `pdksh` is freely available but does not have the full set of functionnality of the AT&T version.

(5) Only by specifying a file via the ENV environment variable.

(6) Can be done by setting shell variables.

B Bibliography

This study was done mainly with:

- Brian Blackmore, The University of Kent at Canterbury, UNIX shell differences and how to change your shell in comp.unix.shell.
- Karsten Kuenne, a report at DESY DUX meeting about UNIX shells.
- Tom Christiansen, faq, `csh-whynot`