• UNIX shells

by Karsten Künne

The first environment you interact with when you log in to a UNIX system is the shell. The shell is the interface between you and the system which accepts your input and executes the appropriate commands. One of the big advantages of UNIX is that you can choose between several different shells, you could even write your very own private shell and use it.

In order to choose an appropriate working environment for myself, I compared various widely available shells.
The candidates were:

sh	(Bourne Shell) The Grand-daddy of all shells, the first UNIX shell, written by Steve Bourne.
csh	The shell from the BSD UNIX Distribution, written at the University of California, Berkeley.
ksh	(Korn Shell) A successor of the Bourne Shell from AT&T, written by David Korn.
tcsh	A very popular "free" shell.
bash	(Bourne Again Shell) The shell from the GNU Project UNIX, also a free shell.
zsh	Another very good free shell, written by Paul Falstad.
rc	The shell for the Plan 9 Project, a possible successor of UNIX, written by Tom Duff at AT&T.

From the syntax point of view most shells fall into one of two main groups: shells with sh-like syntax and shells with csh-like syntax. sh-like syntax and csh-like syntax are not compatible. The first group contains sh, ksh, bash and zsh whereas the second group contains csh and tcsh. The rc shell has a different syntax which does not fit into either of the groups.

What a shell should do

What can one expect from a shell? What are the main requirements an interactive shell has to meet?:

1. **Execution of commands:** The shell should execute commands.
2. **Pipes:** The ability to concatenate commands such that the output of one command is used as input for the next.
3. **Redirection:** This means redirecting command input/output from/to files instead of standard input (keyboard) and output (screen).
4. **Configurable Environment:** The ability to define a personal working environment using variables.
5. **History:** A history of executed commands and the possibility to re-execute commands.
6. **Globbing:** The expansion of wild-cards in filenames.
7. **Aliases:** The ability to use aliases for commands.
8. **Functions:** The ability to use shell functions.
9. **Command Line Editing** The ability to edit a command line before execution using the arrow keys.
10. **Job Control:** Report of status changes for background jobs and track-keeping of background jobs.
11. **Completion:** The completion of a partially typed command or file name.
12. **Spelling Correction:** The correction of spelling errors in commands and file names.

Additional demands exist especially for the non-interactive use of a shell (i.e. for shell scripts):

13. **Execution Control:** Loops and conditional expressions are required.
14. **Signal Handling:** The ability to establish handlers to catch signals and perform specific actions.
15. **Arithmetic:** The ability to do arithmetic computations.

Free Shells

The vendor supplied shells sh, csh and ksh should be well known to most UNIX users, but how do the free shells compare to these standard shells?

tcsh

The tcsh is basically a csh with a lot of enhancements. It behaves exactly like the csh, except for the added utilities. The most important of these utilities are :

- Command line editing using Emacs-style commands.
- Visual step up/down and searching through the command history list.
- Programmable command, file name, variable name, and user name completion.
- One can give a command to produce a file/directory/user list in the middle of a typed command.
- Spelling correction of command, file, and user names.
- Enhanced history mechanism.
- An addition to the syntax of filenames to access entries in the directory stack.
- In file expressions one can specify negation of a set of characters or a globbing pattern.

There are more enhancements built into tcsh but these are the most important ones.

bash

The bash is basically a sh with a lot of enhancements and features from (t)csh and ksh and some bash specific features. The bash specific features are :

- Command line editing with arrow keys, configurable.
- Commands `set` and `help` available.
- Additional startup files and shell variables.

Apart from this bash is comparable to ksh.

zsh

The zsh is a very powerful shell. The grammar of zsh is very close to ksh/sh, with csh additions. Zsh contains most features of ksh, bash, and tcsh. Some of the important zsh features are :

- A shorthand for loops. Example:
  `for i (*.c) echo $i`
- A directory stack exists that is accessible with the `dirs` command and `=number`.
- Process substitution. Example:
  `vi` $=(cmd)$
  starts `vi` on the output of cmd
- Generalized pipes. Example:

  `ls foo >>(`$cmd1$`) 2>>(`$cmd2$`)`

  pipes `stdout` to cmd1 and `stderr` to cmd2.
- Advanced globbing. Examples:

  `ls **/file`

  searches recursively for "file" in subdirectories.

  `ls file<20->`

  matches file20, file30, file100 etc.

  `ls *.(c|pro)`

  matches *.c and *.pro.

  `ls *(R)`

  matches only world readable files.

  `ls *.c~lex.c`

  matches all .c files except lex.c.
- Null command short-hands. Examples:

`< file`	is the same as	`more < file`
`> file`	is the same as	`cat > file`
`>> file`	is the same as	`cat >> file`

- Automatic file stream teeing (redirect to two different output files). Example:

  `ls >foo >bar`

  puts output in two places.

- Incremental history search.

- With the `autocd` option, typing a directory name by itself is the same as typing `cd dirname`.

- Menu completion. Pressing TAB repeatedly cycles through the possible matches.

- Incremental path hashing.

- With `histverify` option, performing csh-style history expansions causes the input line to be brought up for editing instead of being executed.

- Auto-loaded functions (loaded from a file when they are first referenced).

- Generalized argument completion including command name completion, filename and path completion, hostname completion, key binding completion, option completion, variable name completion, and user-specified keyword completion.

- Various nested startup files.

- "which –a cmd" lists all occurrences of "cmd" in path.

rc

The rc shell is a bit exotic compared to the other shells. The syntax is similar to sh, but more based on awk and C. The most important new features compared to sh are:

- Semantic simplifications.

- Parser is based on yacc. This leads to an exact grammar.

- One pass scanning of input stream.

- Signal handling through functions with the signal name.

- Advanced redirection. Examples:

  `cmd >[2] ...`

  redirect file descriptor 2.

  `cmd >[2=1] ...`

  replace file descriptor 2 by a copy of descriptor 1

  `cmd |[2] ...`

  pipe file descriptor 2 into another command.

- Simpler command substitution Example:

  `echo '{`cmd`}`

  substitutes output of cmd as parameter for echo.

- Array variables.

- Concatenation operator. Example:

  `echo (a b c)^(1 2 3)` is the same as
  `echo a1 b2 c3`

Comparison

For a short comparison of the different shells the following matrix gives a good overview. The numbers on the left side are the numbers of the requirements at the beginning of this article.

	sh	csh	ksh	tcsh	bash	zsh	rc
1	+	+	+	+	+	+	+
2	+	+	+	+	+	++	++
3	+	–	+	–	+	++	++
4	+	+	+	+	+	+	+
5	– –	+	+	+	+	+	+
6	+	+	+	+	+	++	+
7	– –	+	+	+	+	+	– –
8	+	– –	+	– –	+	+	+
9	– –	– –	+	++	++	++	++
10	– –	+	+	+	+	+	– –
11	– –	– –	+	++	+	++	– –
12	– –	– –	– –	+	– –	+	– –
13	+	+	+	+	+	+	+
14	+	–	+	–	+	+	+
15	– –	+	+	+	+	+	+

++ means "very good support"

+ means "good support"

– means "limited support"

– – means "no support"

Conclusion

As one can see from the comparison above, some excellent "free" shells exist besides the vendor supplied ones which compete very well. In general these "free" shells have a lot of enhancements which can really improve the daily work with a UNIX system and could increase the general acceptance level of UNIX.

For me the absolute winner in the comparison is zsh. It's really the most powerful shell I have ever used. I had tried tcsh and bash before, but now I would recommend zsh. I have now been using zsh for several months and have no problems at all. Some of the zsh features are really nice and made my life with UNIX easier.