Shell History: Unix

Written by Christian Lee Seibold. September 2-4, 2020.

A Shell is a textual interface that allows you to type commands to do various things, including running programs and running files that list a sequence of commands, initially called “runcoms”, and later, shell scripts. Shells seem to stem from one operating system, MIT’s CTSS, which was very influential in both the Unix line, and the DOS/Windows line of Operating Systems. As we will see, MIT, DEC, and Bell Labs influenced the computer space deeply. In this article, we discuss the Unix line, starting with RUNCOM and Multics, a Project MAC Operating System with the goal of being a successor to CTSS.

RUNCOM and Multics

Unix Shells have had a very long history, and it all starts with a program written by Louis Pouzin for the MIT CTSS Operating System, called RUNCOM (which stood for “run commands”). It executed commands from a file, called “a runcom”. According to Kernighan and Ritchie[1], “rc” configuration files from Unix descended from this. Tom Van Vleck also gives origins of Unix’s use of “rc” to RUNCOM [2], and notes that the first time he read the term “shell” was from Multics documentation created by Doug Eastwood. According to Louis Pouzin, he coined the word “shell”. [3]

Multics started development in 1964 as a project of MIT’s Project MAC, in collaboration with GE and Bell Labs. During the year of 1964, Christopher Strachey visited MIT. Louis Pouzin thought Strachey’s macro-generator design, particularly the “techniques for quoting and passing arguments”, was a good base for a command language. Pouzin wrote a paper and a flowchart designing what would become Multics’ “shell” (a term he coined) just before he left. After leaving, this shell was implemented into Multics by Glenda Schroeder and another programmer from GE. [3]

This early implementation of the shell included the basic syntax of `command arg1 arg2` with arguments separated by spaces and terminated by semicolons, strings as return values, command substitution, and iteration, which allowed you to write multiple elements in parentheses and the shell would run each version of the command with each element. So, `print (a b c).epl` became `print a.epl; print b.epl; print c.epl`. It also allowed you to evaluate commands differently, where `[cmd]` would substitute the returned string into the command (called an *active command*), `|[cmd]` would substitute the returned string into the command as another command substitution (called a *neutral command*), and `||[cmd]` would not substitute anything into the command (called an *empty command*). [4]

Pipe syntax was later added to Multics in the 80s. According to Vleck, `io_call` and stream manipulation could achieve the same effect as Unix’s pipes, just without the convenient pipe syntax. [5] However, according to Dennis Ritchie, he doesn’t “think this is true, or is true only in a weak sense” because “Multics spliceable IO modules required that the modules be specially coded in such a way that they could be used for no other purpose”. However, he does note that it did have a “general IO redirection mechanism … embodying the name IO streams”, but that the notation was “very clumsy”. [6]

Multics also had a search path that was used to search for commands that could be used within the shell.

Unix v1-v6 (Thompson) Shell

Due to dissatisfaction with the Multics system, Bell Labs withdrew from the project In 1969. [17] Plans and development for a new Operating System for the PDP-7 was started in 1969. Much of this work, including the command interpreter (shell), was done by Ken Thompson. This shell eventually had IO redirection, inspired by Multics' IO streams. In 1970, Bell Labs purchased a PDP-11, and work on a port of the Operating system to the PDP-11 was started. 1970 was also the year the name "Unix" was proposed. Later, in 1972, pipes, a "specific form of coroutine", were added to the shell, by a suggestion from M. D. McIlroy [6], who invented the concept years prior, in 1964 (p. 68) [17]. While this was not known at the time, Dartmouth Time-Sharing System (DTSS) also had a very similar concept to pipes. At first, pipes were used by stacking up the commands one after another. However, later on, the syntax was switched so that the `>` character was used. [6] According to both John Mashey[8] and David Korn[9], this shell also had `goto` and `if` commands.

While Multics originally had the search path, Unix decided to give up this idea initially. According to Doug McIlroy, the search path was added in v3 of Unix. [18] However, according to Brian Kernighan, the search path was added in PWB Shell. (p. 132) [17]

Mashey/PWB Shell

Work on a replacement shell for Unix was started in mid-1975, according to Stephen Bourne. [7] This shell was written by John Mashey and a group of other people, including Alan Glasser, and was distributed as part of the Programmer's Workbench UNIX. New commands and features were added to Unix's shell, including `switch`, `while`, and variables, 3 of which were derived from per-process data. Some of the existing commands were also improved, including `if/else/endif`. [8] PWB Shell also included a search path so that a sequence of directories could be searched for commands, which was later copied by Bourne Shell. [17]

Bourne Shell

In 1975, it was decided that the shell for Unix should be rewritten to fix issues. At this time, Ken Thompson went off to Berkeley for a year, so the shell was mainly written by Stephen Bourne at Bell Labs. The first version of the shell was deployed in 1976. [7]

Much of the Bourne Shell's syntax was inspired by Algol68. The shell added shell scripts, multi-character variables, here docs, command substitution, path searching, interruptible wait, pattern matching, and string quoting. There were no length restrictions on strings, goto was removed, there were no comments, environment variables were added later, in 1978 [7], and functions, `echo`, and `pwd` were added in 1982, as part of Unix System V [11]. Bill Joy suggested to Bourne that Job Control and History should be added to the shell, to which Bourne disagreed. [7]

Unix version 7, with Bourne Shell as default, was released in 1979.

C Shell (csh)

The C Shell was started in 1978 by Bill Joy at Berkeley. According to David Korn, C shell introduced command history and an editing functionality. [9] In the intro to *The UNIX C Shell Field Guide*, Bill Joy mentions the shell's original ideas were the *history mechanism*, which was inspired by "the history and DWIM features of Interlisp" (invented by Warren Tietelman), *aliasing*, which was patterned "roughly on Lisp reader macros", and *job control*, which was "added … by Jim Kulp". [10]

C shell also had `&` for background commands, here docs, CDPath, directory stacks, path hashing, brace expansion (aka. alternation - used in pattern matching), expression evaluation, and command groups. On page 73, the field guide mentions pathname variables, which allowed you to pick specific parts of a path stored inside a variable using the syntax: `$p:x`, where `x` can be `r` for root, `h` for header, `t` for tail, and `e` for extension.

On page 41 of the Korn Shell manual, it is mentioned that Tilde Notation came from C Shell. [11]

Korn Shell (ksh)

The Korn Shell was developed by David Korn at Bell Labs and was released in 1983. David Korn created the precursor to the Korn Shell as a “form interpreter” by modifying the Bourne Shell, allowing built-in commands to be used in I/O redirection, and adding the `echo`, `pwd`, and `test` built-in commands. [9]

David Korn later implemented the first version of Korn Shell prior to the UNIX System V shell. This first version took history, aliases, and job control from C Shell. [9] According to the Korn Shell Manual, tilde notation, Job Control, directory stack, and the `logout` command all came from C shell (pages 41, 97, 273, and 267 respectively). [11]

Korn Shell also had vi line editing mode, written by Pat Sullivan, and emacs line editing mode, written by Mike Veach. [9]

The 1988 version of Korn Shell extended pattern matching to be similar to regexes found in sed and grep at the time. This version was used as the basis for the POSIX.2 standard in 1992, aka. “IEEE POSIX 1003.2” and “ISO/IEC 9945-2”, which created POSIX standards for shells and utilities. [9]

The Korn Shell Manual [11], from 1992, also mentions the following features:

- Integer Arithmetic (p. 107)
- `<>` to open file as read and write (p. 19)
- Arrays, data types, and variable attributes (p. 43)

TENEX C Shell (tcsh)

TENEX C Shell was developed by Ken Greer. Tcsh was based on C Shell, but added additional features, including “command and filename recognition and completion” written by Mike Ellis, and inspired by the TENEX OS by BBN. [12]

Almquist Shell (ash)

Kenneth Almquist first released Almquist Shell in 1989. It was a reimplementation of the System V shell with added features, including:

- Local variables inside functions
- Function definitions override built-in commands
- Pattern negation
- Job Control from “Berkeley Shell” (C Shell)

He also originally left out aliases and history. [13]

Almquist was used by Android until version 4, when it switched to mksh, a derivative of Korn Shell.

Bourne-Again Shell (bash)

The Bourne-Again shell was developed by Brian Fox for the GNU Project. It was first released in 1989. Chet Ramey took over development in 1994. It was based on the Bourne Shell. It had brace expansion (from C Shell), programmable command line completion, command line editing, command history, the directory stack, functions, arrays, integer arithmetic via the `(())` syntax, and POSIX command substitution via the `$()` syntax.

In 2004, bash introduced associative arrays.

RC Shell

RC was the Shell for Plan 9 and Unix version 10, and was developed by Tom Duff. It replaced the Algol68 like control structures of Bourne with C-like ones. Variables are a list of strings instead of one string, where the list is created by splitting arguments by spaces. This allowed you to reference a specific argument within a variable. The exit status of commands was a character string describing the error. Curly brackets were used for a sequence of commands.

Z Shell (zsh)

Z Shell was created by Paul Falstad and released in 1990. It had editing of multi-line commands, spelling completion/auto-correction and auto-fill of command names, and themeable prompts. [14] It also had termcap support and login/logout watching. [15]

It now has various compatibility modes, loadable modules, and named directories.

POSIX.2 Shell and Utilities Standard

The POSIX.2, Shell and Utilities, Command Interpreter (IEEE Std. 1003.2-1992) standard specified what a POSIX-compliant shell should include. These specifications were based on the System V Shell (Bourne Shell) , “with enhancements from Korn Shell”. [16]

Debian Almquist Shell (dash)

Debian Almquist Shell was a port of Almquist Shell from NetBSD to Debian Linux by Herbert Xu, in 1997. Its goals were POSIX conformance and slim implementation. Internationalization, localization, and multi-byte character encodings, however, were not implemented. It also had optional history and GNU readline line editing support.

Ubuntu adopted dash in 2006. Debian adopted it in version 6 (Debian Squeeze). Although both Ubuntu and Debian adopted dash, bash still remains the default login shell for interactive use.

Conclusion

Today, dash, bash, and Bourne are the shells still in use in the Linux and macOS world. Mksh is a descendant of Korn Shell, and is used on Android. FreeBSD uses tcsh. NetBSD uses ash. And finally, OpenBSD uses ksh. All the shells above have had an immense influence in the computer world, even crossing over to Windows. However, there is another story. This other story *also* involves CTSS. However, it quickly diverges, and, in the end, merges into one story. This story is the story of where DOS and Windows’ command line Shells came from. This is the story that will be covered in the next article.

Note: If you find any errors within this article, please email them to me at: krixano@protonmail.com

Feature Origins

This is a list of shell features and which shell implemented it first:

- Basic Syntax - Commands, arguments: Multics
- Strings returned from commands: Multics
- Iteration: Multics
- Command Substitution: Multics
- Path searching: Multics, Thompson Shell, Mashey/PWB Shell (?)
- Pipes: Thompson Shell, DTSS
- IO Redirection: Multics, Thompson Shell
- goto, basic control flow: Thompson Shell
- switch and while: Mashey/PWB Shell
- Variables: Mashey/PWB Shell
- Shell Scripts: Bourne Shell
- Multi-character variables: Bourne Shell
- Here-docs: Bourne Shell
- Environment Variables: Bourne Shell
- String quoting: Bourne Shell (?)
- Interruptible Wait: Bourne Shell
- Command History: C Shell
- Line Editing: C Shell
- Aliases: C Shell
- Job Control: C Shell
- Directory Stack: C Shell
- Path Hashing: C Shell
- Expression Evaluation built-in: C Shell
- Pathname variables: C Shell
- Tilde Notation: C Shell
- Brace Expansion / Alternation: C Shell
- I/O Redirection for built-ins: Korn Shell
- vi and emacs line editing modes: Korn Shell
- Integer Arithmetic built-in: Korn Shell
- Arrays, data types, and variable attributes: Korn Shell
- Command and Filename Completion: TENEX C Shell (?)
- Local variables inside functions: Almquist Shell
- Variables as list of string arguments: RC Shell
- Shared Command History: Z Shell
- Loadable modules: Z Shell
- Named Directories: Z Shell
- Spelling Auto-correction: Z Shell

Sources

[1] https://kb.iu.edu/d/abnd

[2] Unix and Multics, Tom Van Vleck: https://www.multicians.org/unix.html

[3] The Origin of the Shell, Louis Pouzin: https://multicians.org/shell.html

[4] MULTICS SYSTEM-PROGRAMMERS' MANUAL Section BX.1.00 - Multics Command Language: https://multicians.org/mspm-bx-1-00.html

[5] Glossary of Multics acronyms and terms, Tom Van Vleck: https://multicians.org/mgp.html#pipes

[6] The Evolution of the Unix Time-sharing System, Dennis Ritchie: https://www.bell-labs.com/usr/dmr/www/hist.html

[7] Early days of Unix and design of sh, Stephen Bourne: https://www.youtube.com/watch?v=2kEJoWfobpA

[8] Mashey Shell Repost, John Mashey: https://www.in-ulm.de/~mascheck/bourne/n.u-w.mashey.html

[9] ksh - An Extensible High Level Language, David Korn: https://www.in-ulm.de/~mascheck/bourne/korn.html

[10] The UNIX C Shell Field Guide, Gail and Paul Anderson: https://archive.org/details/unixcshellfieldg00ande/page/326/mode/2up

[11] The Korn Shell User and Programming Manual, Anatole Olczak: https://archive.org/details/kornshelluserpro1992olcz/page/368/mode/2up

[12] https://groups.google.com/g/net.sources/c/BC0V7oosT8k/m/MKNdzEG_c3AJ?pli=1

[13] A reimplementation of the System V shell: https://groups.google.com/g/comp.sources.unix/c/A6cnyKX-Gq4/discussion

[14] zsh - a ksh/tcsh-like shell (part 1 of 8): https://groups.google.com/g/alt.sources/c/tVgN49u8Ax4/m/7VgQlHZ4bJMJ

[15] zsh - a ksh/tcsh-like shell (part 2 of 8): https://groups.google.com/g/alt.sources/c/NiVcvSh3P04/m/tY4ZswxbOUsJ

[16] Jesperson, Hal. "Emerging standards." *UNIX Review*, vol. 11, no. 3, Mar. 1993, p. 30+. *Gale Academic OneFile*, https://link-gale-com.briarcliff.idm.oclc.org/apps/doc/A13761630/AONE?u=briarcliffu&sid=AONE&xid=575043a6. Accessed 31 Aug. 2020.

[17] Brian W. Kernighan. *UNIX: A History and Memoir*. Kindle Direct Publishing, 2020.