****

**A Project Report on**

**“Features Common to the Bourne Again and TC Shells”**

**Submitted to Lovely Professional University**

**“M.C.A.”** MASTER OF COMPUTER APPLICATION

**Submitted By-**

**Name: SAURABH KUMAR Roll No.: A05**

**Registration No.: 12212496 Section: D2217**

**Under The Guidence & Supervision of**

**Dr. Devender Kumar**

**Assistant Professor**

**Abstract**

**The Bourne Again Shell and TC Shell are command interpreters and high-level programming languages. As command interpreters, they process commands you enter on the command line in response to a prompt. This chapter focuses primarily on the Bourne Again Shell, while noting where it differs from the TC Shell.**

**Introduction**

**Similar to how history is used in bash, so is it in tcsh.  Both shells function with the same event and word designators. For instance, much like in bash,!! in tcsh refers to the preceding event. Lists the few extra word modifiers in tcsh not present in zsh or bash. The command!328 means to execute event number 328 and!?txt? means to execute the most recent event containing the string txt.**

u Converts the first lowercase letter into uppercase

l Converts the first uppercase letter into lowercase

a Applies the next modifier globally within a single word



**It is possible to use more than one word modifier in a command. For instance, the a modifier, in combination with the u or l modifier, is handy for changing the case of an entire word**

tcsh $ echo $VERSION

VERSION: Undefined variable.

tcsh $ echo !!:1:al

echo $version

tcsh 6.12.00 (Astron) 2002-07-23 (i386-intel-linux) options 8b,nls,...

**The variables that you set to control history in tcsh are different from those in bash. Where bash uses HISTSIZE and HISTFILESIZE to determine the number of events that are preserved during and between sessions (page 590), tcsh uses history and savehist**

|  |  |  |
| --- | --- | --- |
| **Function** | **Variable** | **Default** |
| **Maximum number of events saved during a session** | **history** | **100 events** |
| **Location of the history file** | **histfile** | **~/.history** |
| **Maximum number of events saved between sessions** | **savehist** |  |

**When you exit from the shell, the most recently executed commands are saved in the ~/.history file. Next time you start the shell, this file initializes the history list. The value of the savehist variable determines the number of lines of history saved in the ~/.history file (not necessarily the same as the history variable). The history variable holds the number of events remembered during a session, the savehist variable holds the number remembered between sessions.**

The TC Shell assigns a sequential event number to each command line. You can display this event number as part of the tcsh prompt . Examples in this section show numbered prompts when they help to illustrate the behavior of a command or group of commands.

If you set the value of history too high, it can use too much memory. If it is unset or set to zero, the shell does not save any commands. To establish a history list of the 100 most recent events, give the following command manually, or place it in your .tcshrc startup file:

tcsh $ set history = 100

The following command causes tcsh to save the 100 most recent events across login sessions:

|  |
| --- |
| **tcsh $ set savehist = 50** |

You can make multiple assignments within a single command. Combining the two preceding assignments gives the same result. (The SPACEs or absence thereof around the equal signs is not significant.)

|  |
| --- |
| **tcsh $ set history=100 savehist=50** |

The 50 most recent events from the previous login sessions remain in your history list when you log out and back in after setting savehist. If you want to keep your event list consistent from login to login, enable savehist in your.tcshrc file.

If set, the history literal variable histlit (history literal) displays the commands in the history list verbatim, with no shell interpretation. This variable's impact is demonstrated in the example that follows (compare lines 32), along with that of the -T option, which forces history to provide timestamps for each command:

|  |
| --- |
| tcsh $ cat /etc/csh.cshrc  . . .  tcsh $ cp !!:1 ~  cp /etc/csh.cshrc ~  tcsh $ set histlit  tcsh $ history -T  . . .  31 9:35 cat /etc/csh.cshrc  32 9:35 cp !!:1 ~  33 9:35 set histlit  34 9:35 history -T  tcsh $ unset histlit  tcsh $ history -T  . . .  31 9:35 cat /etc/csh.cshrc  32 9:35 cp /etc/csh.cshrc ~  33 9:35 set histlit  34 9:35 history -T  35 9:35 unset histlit  36 9:36 history -T |

Use the history builtin to display the events in your history list. The list of events is ordered with the oldest events at the top. The last event in the history list is the history command that displayed the list. The following history list includes a command to modify the tcsh prompt to display the history event number and the command number. To simplify the example, history has been set to 10 and savehist to 20. (The event number is 20 greater than the command number because the list of events includes those events that were saved from the last login session—20 in this case.)

|  |
| --- |
| **32 12 $ history**  **23 set prompt = "! $ "**  **24 ls -l**  **25 cat temp**  **26 rm temp**  **27 vi memo**  **28 lp memo**  **29 vi memo**  **30 lp memo**  **31 rm memo**  **32 history** |

As you run commands and your history list becomes longer, history produces a list that runs off the top of the screen. Use a pipe to send the output of history through less to browse through it or give the command history 10 to look at your ten most recent commands.

You can change the name of the history file (normally .history) by changing the value of the histfile variable:

|  |
| --- |
| **tcsh $ set histfile = "~/.tcsh\_dir/history"** |

The histfile variable is not present in the TC Shell.

In addition to using event designators to access the history list, you can use the command line editor to access, modify, and execute previous commands

**Optional**

**There is a difference in how bash and tcsh expand history event designators. If you give the command !250w, bash replaces it with the command number 250 with a character w appended to it. In contrast, tcsh looks back through your history list for an event that begins with the string 250w to execute. The reason for the difference is that bash interprets the first three characters of 250w as the number of a command, whereas tcsh interprets them as part of the search string, 250w. (Of course, if the 250 stands alone, tcsh treats it as a command number.)**

**If you want to append a w to command number 250, you can insulate the event number from the w by surrounding it with braces:**

|  |
| --- |
| **!{250}w** |

**Alias**

**The alias/unalias feature in tcsh closely resembles its counterpart in bash. The alias builtin itself, however, has a slightly different format. In bash you can create an alias for ls with the following command:**

|  |
| --- |
| **bash $ alias ls="ls –lF"** |

**In tcsh you can create the same alias by replacing the equal sign with a SPACE:**

|  |
| --- |
| **tcsh $ alias ls "ls –lF"** |

**Some alias names, called special aliases, have special meaning to tcsh. If you define an alias with one of these names, it executes automatically at certain points in your interaction with the shell. Initially all the special aliases are undefined.**

|  |  |
| --- | --- |
| **Special Alias** | **Executed** |
| **beepcmd** | Whenever the shell would normally ring the terminal bell. This gives you a way to have other visual or audio effects take place at those times. |
| **cwdcmd** | Whenever you change to another working directory. |
| **periodic** | Periodically, as determined by the number of minutes in the tperiod variable. If tperiod is unset or has the value 0, you cannot set periodic |
| **precmd** | Just before the shell displays a prompt. |
| **shell** | Gives the name of the interpreter that you want to use on scripts that do not start with #! . The first word of the alias must be the full pathname of the interpreter to be used. |

To see a list of the current aliases, give the command alias. To view the alias for a particular name, give the command alias followed by the name.

**Differences between the tcsh and bash alias Mechanisms**

**The alias builtin and substitution used in bash are patterned after the alias builtin found in the Z Shell. It is slightly different from the alias builtin used in tcsh, which is patterned after the alias builtin found in the TC Shell. The syntax of the version used by tcsh is**

|  |
| --- |
| **alias name value** |

**The tcsh version of alias (next section) lets you substitute the command arguments. To do something similar in bash, you have to use a shell function**

**History Substitution in an Alias**

**You can substitute command line arguments by using the history mechanism, with a single exclamation point representing the input line containing the alias. Modifiers are the same as those used by history . The exclamation points are quoted in the following example so that the shell does not interpret them when building the aliases (which would produce incorrect results):**

|  |
| --- |
| **21% alias last echo \!:$**  **22% last this is just a test**  **test**  **23% alias fn2 echo \!:2:t**  **24% fn2 /home/jenny/test /home/alex/temp /home/barbara/new**  **temp** |

**Event 21 defines for last an alias that displays the last argument. Event 23 defines for fn2 an alias that displays the simple filename, or tail, of the second argument on the command line.**

**Job Control**

**Job control in bash and in tcsh is similar. You can move commands between the foreground and background, suspend jobs temporarily, and get a list of the current jobs. The % character references a job when followed by a job number or a string prefix that uniquely identifies the job. You will see a minor difference when you run a multiple-process command line in the background. Whereas bash displays only the PID number of the last background process in each job, tcsh displays the numbers for all the processes belonging to a job**

|  |
| --- |
| **tcsh $ find . –print | sort | lpr & grep –l alex /tmp/\* > alexfiles &**  **[1] 18839 18840 18841**  **[2] 18876** |

**Filename Substitution**

**Similar to bash, the TC Shell extends the characters \*,?, and [] in a pathname. The symbols \* and? match strings of zero or more characters, respectively, while [] provides a character class that is used to match single characters found between two brackets.**

**Similar to bash, the TC Shell converts command line arguments that begin with a tilde () into filenames, where the tilde stands for the user's home directory or the user whose name follows the tilde. (Tcsh does not support the special expansions + and -.)**

**Brace expansion is a feature of filename substitution and is present in tcsh, just like tilde expansion. Despite the fact that brace expansion can result in strings that.**

**Manipulating the Directory Stack**

**Directory stack manipulation in bash and in tcsh does not differ much. The dirs builtin displays the contents of the stack, and the pushd and popd builtins push directories onto and pop directories off the stack.**

**Command Substitution**

**The $(…) format for command substitution is not available in tcsh. In its place you must use the original '…' format. Otherwise, the implementation in bash and tcsh is identical. Refer to for more info on command substitution.**

**Refference**

* [**https://learning.oreilly.com/library/view/a-practical-guide/0201703130/0201703130\_ch14lev1sec3.html**](https://learning.oreilly.com/library/view/a-practical-guide/0201703130/0201703130_ch14lev1sec3.html)
* [**https://flylib.com/books/en/3.162.1.112/1/**](https://flylib.com/books/en/3.162.1.112/1/)