Original scientific paper

An Optimization of Command History Search

ata, citation and similar papers at core.ac.uk
brought to yo
provided by Repozitorij Filozofskog fakulteta u Zagrebu' at Uni

Faculty of Humanities and Social Sciences
Ivana Lučića 3, Zagreb, Croatia
vladimir.mateljan@gmail.com

Krunoslav Peter
Faculty of Humanities and Social Sciences
Ivana Lučića 3, Zagreb, Croatia
kruno_peter@yahoo.com

Vedran Juričić
Faculty of Humanities and Social Sciences
Ivana Lučića 3, Zagreb, Croatia
vedran.juricic@gmail.com

Summary

A command that a user issues at the command prompt is a string encoded with certain character encoding. This character string can be stored into an appropriate data structure in order to be documented or reused. Additional functionality of the command line to store entered commands, along with ability to list, edit and re-execute previously entered commands, is called command history. The paper suggests an optimization of command history search based on proposed grammar of the command language.

Key words: command, command line, formal language, string, command history, grammar, algorithm, search, optimization, grep, awk

Introduction

This paper focuses on commands as words of a formal language and the search for those commands in command history. A grammar of the command language is proposed; this grammar defines the command as a word of a formal language that consists of singleton words named "single-words," separated by spaces. A general search algorithm reads commands from command history and matches commands that contain search string. Four examples of command history implementations, in various operating systems, follow the description of the command language. There are also two examples of command history search by using software tools. The problem is how to find only commands that contain single-words equal to search string. An optimization of command history search is based on abstraction "a command as a sequence of single-words, separated by

spaces." Modified general search algorithm reads command from command history, parses the command string to divide it up into single-words and then matches any single-word to find one that is equal to search string.

The command line

A *command line*, as a way of interacting with software system, is provided by operating systems shells, programming languages that have interactive mode, query processors of the database management systems, or other interactive software systems. The command *prompt* is a visual component of the command line[1]; it tells the user that he can enter a command. A blinking *cursor* following the command prompt indicates the place where next entered *character* will appear (Example 1).

The command line has the *syntax* and *semantics*. In short, the syntax defines rules for writing commands; the semantics defines what commands do. A *command interpreter* parses a command issued by the user and coordinates what happens between the user and the software system (Peek et al, 1998, 5).

Example 1: The command line with the prompt ('**#**'), the cursor ('**_**'), and the command *ls*[2] ('`ls -al`')

```
# ls -al_
```

An additional functionality of command line called the *command history* is to keep in appropriate data structure a history of executed commands along with ability to list, edit and re-execute previously entered commands.

The command as a word of a formal language

The command is the *word* of a formal language; it is a *string* of characters (Klint, 1985, 5) over an *alphabet* that a user types in the command line. Let A be a finite set called the alphabet (Abiteboul, 1995, 13); the command c over an alphabet A is a finite sequence $a_1...a_n$, where $a_i \in A$, $1 \leq i \leq n$, $n \geq 0$. Its length is n. Empty string is denoted by ε and its length is 0 (Dovedan, 2003, 15). The alphabet is implemented as the *character encoding* in software system and the string is a finite sequence of characters.

[1] The command line can also be implemented as an input field in the graphic user interface.

[2] The *ls* command is well known command of the operating system Unix that lists the content of a directory; option '*–a'* is used to list the content for a directory and '*–l'* to create a vertical list of a directory's content (Toporek, 2003, 87).

When the user enters a command, he performs the *concatenation* of characters (or strings). "If x and y are strings, then the concatenation of x and y, written xy, is the string formed by appending y to x (Aho et al, 1986, 92). The concatenation and substring selection are the most primitive operations on strings (Klint, 1985, 5).

If x, y, and z are words over alphabet A, a concatenation of this words is a word xyz. Strings x, y, z, xy, and yz are *substrings* of string xyz. Concept of the substring is important for the focus of this paper, because the command history can be searched for commands that contain a certain substring.

The set of all words over alphabet A is denoted by A^* (Abiteboul, 1995, 13). The *formal language* L is a subset of A^* ($L \subseteq A^*$). A *command language* is also a formal language over an alphabet.

The *grammar* is a formal system for generating words of a formal language (Dovedan, 2003, 19). A *context-free grammar* G_c that defines the command language $L(G_c)$, is a 4-tuple $G_c(N, A, S, P)$, where:

- N is a finite set of *non-terminal* symbols *{Z, W}*;
- A is a finite set of *terminal symbols* (characters), disjoint from N, an alphabet that contains lowercase letters *{a, b, c, d,...}*, numbers *{0, 1, 2, 3,...}*, space *{' '}*, and other symbols *{'.', '|', '>', ...}*;
- S is a *start symbol*, a distinguished symbol from N;
- P is a finite set of *productions* of the form $\alpha \rightarrow \beta$, called Backus-Naur form (BNF) (Dovedan, 2003, 35), where is $\alpha \in N$ and $\beta \in (N \cup T)^*$:

$S \rightarrow \varepsilon$
$Z \rightarrow$ ' ' *(space)*
$W \rightarrow$ *{a | b | c | ... | 0 | 1 | 2 | ... | .| > | $ | * | ...}*
$S \rightarrow \varepsilon \mid W \mid W$ *[{{Z}W}]*

Non-terminal symbol Z represents white space (blank). W stands for *single-word* – a sequence of characters without any spaces. *{Z}W* is the concatenation of one or more spaces and a single-word.

Presented grammar G_c of the command language $L(G_c)$ defines the language consisting of all words in A^* that can be derived from the start symbol S by repeated applications of the productions; a word of $L(G_c)$ can be:

- an empty string (ε),
- a single-word (W),
- or a string of single-words separated by one or more spaces (W *[{{Z}W}]*).

Words of the command language $L(G_c)$ correspond to the commands of the operating systems *Unix*, *DOS*[3], *Microsoft Windows XP* and *Vista* (Example 2). Operating system's commands can be simple, single-word entries or more complex. The general syntax for commands is (Mateljan et al, 2007, 448):

command [{option}] [{filename}]

So, by using proposed grammar G_c, commands can be generated as words of a formal language where single-word is:

- a command (without any arguments, options, piping, and redirection),
- an optional argument or of the command (e. g. a filename (without any spaces), a name of the variable, or a name of the constant),
- an option of the command (e. g. '`/p`' or '`-l`'),
- an operator (e. g. redirection operator '>' or adding operator '+').

Example 2: Commands *dir* and *more*[4] of the operating systems DOS and Microsoft Windows XP/Vista

```
dir
dir *.txt
dir | more
dir *.txt > dir.txt
```

Words of $L(G_c)$ also conform to the commands of the programming language *Tcl*[5] (Example 3). Tcl has interactive mode that gives a user an ability to learn individual Tcl commands. A *Tcl script* consists of one or more commands. Each command consists of one or more single-words, where the first single-word is the name of the command and additional words are arguments to that command. Single-words are separated by spaces or tabs (Ousterhout, 1994, 29). So, the general syntax for Tcl commands is:

command [{argument}]

[3] *DOS* (short for Disk Operating System) is an operating system copyrighted by Microsoft in 1979 and initially written by Tim Paterson. There are several similar products produced by other companies. For example, FreeDOS is a complete, free, MS-DOS compatible operating system. The command interpreter of Windows XP/Vista, *cmd.exe*, maintains most of DOS commands (Wikipedia, 2009).

[4] Well known command *dir* lists the content of a directory; the command *more* displays the output one screen at a time.

[5] *Tcl* (originally from "Tool Command Language") is a scripting language created in the spring of 1988 by John Ousterhout (Wikipedia, 2009).

Example 3: Commands *set* and *expr*[6] of the programming language Tcl

```
set preset1 2
set preset2 3
expr $preset1 + $preset2
```

To search a command history for certain search string, a general search algorithm can read commands from command history and match commands that contain a search string. If a user wants to find only commands that contain single-words equal to search string, he should avoid matching entire command, because the search string can be a substring of the single-word. For example the search string '*set*' is the substring of the variable name '*preset1*'.
Regardless of concrete command language and its lexical structure[7], spaces are inserted between single-words in the command in majority of languages. Hence, in this paper the command is considered as a sequence of single-words separated by spaces. This *abstraction* – "a command as a sequence of single-words separated by spaces, where single-word is a sequence of characters without any spaces" – is a key for the optimization of command history search. So, an *optimization* of the command history search is to find only those commands that contain single-words equal to search string. Instead of matching entire command, the search algorithm parses the command and matches any single-word to find one that is equal to search string. This search can be performed without a need for knowing the semantics of a command language.

Command history

A command that a user executes at the prompt can be stored as a string into an appropriate data structure. It could be a sequential file. This paper doesn't concern itself with the data structures. It is only important that stored commands are linearly ordered by creation in command history and that they could be read one by one.
Even the commands that have syntax errors should be stored to the command history, because the user can easy recall these commands from command history, edit and then execute.
When the commands are stored in the command history, they can be reused in many ways:

- recalling and executing a previous command;
- recalling and editing a previous command in case of syntax error;

[6] The *set* command is used to write and read variables. The *expr* command evaluates an expression; it treats all of its arguments together as an arithmetic expression (Tcl Developer Xchange, 2008).

[7] "Grammars are capable of describing most, but not all, of the syntax of programming languages. A limited amount of syntax analysis is done by a lexical analyzer." (Aho et al, 1986, 172)

- copying a command from command history to another context (batch script or document);
- a statistical analysis (for example, counting the most frequently used commands in order to be replaced by aliases).

In this part of the paper is an overview of four implementations of the command history in various operating systems (Table 1). The shells in the operating systems DOS and Microsoft Windows XP/Vista store the command history in main memory, while the shell of *Apple Mac OS X* "keeps track of recently entered commands" (Toporek, 2003, 74) in the text file *.tcsh_history* that is located in home directory of the user. The *Bash*[8] shell in Unix also stores the command history in the text file called *.bash_history* (Cameron & Rosenblatt, 1998, 30). To recall previously entered command, the user can use "Up Arrow" key (⇧). When the user issues the command '`doskey /h`' in DOS or Windows XP/Vista operating system, he will see the content of the command history. In Mac OS X and Unix, the command for listing the command history is '`history`'.

Table 1: Implementations of the command history in various operating systems

Operating system	DOS	Windows XP/Vista	Mac OS X	Unix (with Bash shell)
Storage	main memory	main memory	file *.tcsh_history*	file *.bash_history*
Recalling previous command	⇧	⇧	⇧	⇧
Command for the list of commands	`doskey /h`	`doskey /h`	`history`	`history`

Command history search

Let $c_1...c_n$, $c_i \in L(G_c)$, $1 \leq i \leq n$ be a sequence of commands in appropriate data structure. The simplest way to find commands that contain search string x is to get commands one by one and put matched commands in output data structure (the pseudo-code of the Algorithm 1):

Algorithm 1: Command history search – matching entire command

Input: a sequence of commands $c_1...c_n$, $c_i \in L(G_c)$, $1 \leq i \leq n$; search string x
Output: a sequence of commands $c_x...c_y$, where c_x is from $c_1...c_n$,..., and c_y is from $c_1...c_n$

[8] *Bash* (short for Bourne Again Shell) is the command interpreter of the operating system Unix (Cameron & Rosenblatt, 1998).

n = "command count";
i = 1;
while (i <= n) do
get c_i from $c_1...c_n$;
if "x is a substring of c_i" then put c_i in $c_x...c_y$;
i = i + 1;
end;

To search the command history in the operating system Unix, a user can use the software tool called *grep*[9]. Example 4 demonstrates command history search by using the utility *grep*.

Example 4: Command history search for substring '*less*' by using *grep*

Typical content of the text file *.bash_history*:

```
ls | less
ls *.txt
ls *.txt > lstxt.txt
less lstxt.txt
rm lstxt.txt
```

If the user wants to find all the commands that contain single-word '*less*', he (or she) should issue this command:

```
# grep less .bash_history
```

The result of command execution will be:

```
ls | less
less lstxt.txt
```

What if the user wants to search for commands that contain the single-word '*ls*'? If he executes the command '`grep ls .bash_history`', the result will be – entire content of the file *.bash_history*. Proposed optimization of command history search, based on abstraction "a command as a sequence of single-words separated by spaces," solves this problem.

[9] The *grep* utility, originally written for Unix, searches a text file for lines that have a certain text pattern (Peek et al, 1998, 71), formally called *regular expressions* (Robbins, 2000, 2). Regular expressions are not covered in this paper. The *find* command in DOS and *findstr* in Windows XP/Vista are similar to *grep*. There are also *grep* executable files for DOS and Windows XP/Vista operating systems.

Optimization of command history search

A better way to search commands that match the string *x* is to get commands one by one, parse the command string to divide it up into single-words, and then match any single-word to find one that is equal to search string *x* (Algorithm 2):

Algorithm 2: Command history search – dividing command up into single-words and matching any single-word

Input: a sequence of commands $c_1...c_n$, $c_i \in L(G_c)$, $1 \leq i \leq n$; *search string x*
Output: a sequence of commands $c_x...c_y$, *where* c_x *is from* $c_1...c_n$,..., *and* c_y *is from* $c_1...c_n$

```
n = "command count";
i = 1;
while (i <= n) do
   get c_i from c_1...c_n;
   founded = 0;
   for each "single-word in c_i" do
      if "x is equal to the single-word" then founded = 1;
   end;
   if (founded = 1) then put c_i in c_x...c_y;
   i = i + 1;
end;
```

In Unix, a user can perform command history search by parsing a command into single-words by using programming language *awk*[10] (Example 5). Each input line (that is a command) of the command history, *awk* will divide into fields (single-words) by "white" space (spaces or tabs); by default, a space is a field separator. "Fields are referred to by the variables *$1*, *$2*, …, *$n*" (Robbins, 2000, 25).

Example 5: Command history search for single-word '*ls*' by using *awk*

The content of Unix file *.bash_history* is shown in Example 4.

The *awk* command, that finds single-word '*ls*' in any command in the file *.bash_history*, is specified on the command line:

[10] Programming language *awk* is designed for processing text-based data, either in files or data streams. It was created at Bell Labs in 1977. The name *awk* is derived from the family names of its authors — Alfred Aho, Peter Weinberger, and Brian Kernighan (Wikipedia, 2009).

```
# awk '{ for (i = 1; i <= NF; i ++) { if ($i == "ls") print
$0 } }' .bash_history
```

Previous command is wrapped-around, because it's too long to be displayed in one row.
This *awk* command corresponds to the statement '*for each "single-word in c_i" do (...)*' of the Algorithm 2. A built-in variable *NF* in the *for*-loop stores the number of fields in current line (Robbins, 2000, 29) or single-words in current command in case of the command language. The *$i* variable represents *i*th field[11] in current line (ibid) or single-word in current command. The *print* command (Mateljan et al, 2007, 449) prints out the commands from command history that contain a single-word equal to search string:

```
ls | less
ls *.txt
ls *.txt > lstxt.txt
```

Previously *awk* command can be written inside a script to avoid retyping (Example 6).

Example 6 – A batch script for command history search in Windows XP/Vista

There are *awk* executables for the operating systems DOS and Windows XP/Vista. So, the output of the command '`doskey /h`' can be piped to the *awk* command. First line of the batch script, called e.g. *chs.bat*, can be the '`@echo off`' command; it turns off the command-echoing.
The *type* command shows the content of batch script *chs.bat* that is located in the root directory of the disk *C*:

```
C:\> type chs.bat
@echo off
doskey /h | awk "{ for (i = 1; i <= NF; i++) { if ($i ==
\"%1\") print $0 }}"
```

Although the second command in the script is wrapped-around in this paper (or on the screen), it is one line in the file *chs.bat*.

[11] As mentioned, *awk* is a language for processing files of text (Robbins, 2000, 23). "A file is treated as a sequence of records, and by default each line is a record. Each line is broken up into a sequence of fields, so we can think of the first word in a line as the first field, the second word as the second field, and so on. An *awk* program is of a sequence of pattern-action statements; *awk* reads the input a line at a time" (Wikipedia, 2009).

The script *chs.bat* has one argument ('`%1`') – a search string. The syntax of command *chs* is:

chs search_string

To find the *more* command in the command history, the user simply types '`chs more`' at the prompt.

Conclusion

An optimization of command history search based on abstraction "a command as a sequence of single-words separated by spaces," allows a user to find only those commands that contain single-words that exactly match the search string, regardless of concrete command language.

References

Abiteboul, Serge; Hull, Richard; Vianu, Victor. Foundation of Databases. Reading : Addison-Wesley, 1995

Aho, Alfred; Sethi, Ravi; Ullman, Jeffrey. Compilers, Principles, Techniques, and Tools. Reading : Addison-Wesley, 1986

Newham, Cameron; Rosenblatt Bill. Learning the Bash Shell. Sebastopol : O'Reilly, 1998

Dovedan, Zdravko. Formalni jezici: sintaksna analiza. Zagreb : Zavod za informacijske studije Odsjeka za informacijske znanosti Filozofskog fakulteta, 2003

Klint, Paul. A Study in String Processing Languages. Berlin Heidelberg : Springer Verlag, 1985.

Mateljan, Vladimir; Požgaj, Željka; Peter Krunoslav. Značaj skriptnih jezika za administraciju operacijskih sustava. // INFuture2007: Digital Information and Heritage / Zagreb : Odsjek za informacijske znanosti Filozofskog fakulteta, 2007, 445-456

Ousterhout, John. Tcl and the Tk Toolkit. Reading : Addison-Wesley, 1994

Peek, Jerry; Tondino, Grace; Strang, John. Learning the UNIX Operating System, 4th Edition. Sebastopol : O'Reilly, 1998

Robbins, Arnold. sed & awk Pocket Reference. Sebastopol : O'Reilly, 2000

Tcl Developer Xchange. Language. 20 October 2008. http://www.tcl.tk/about/language.html (20 August 2009)

Toporek, Chuck. Mac OS X Pocket Guide, 2nd Edition. Sebastopol : O'Reilly, 2003

Wikipedia. awk. 20 August 2009. http://en.wikipedia.org/wiki/AWK (20 August 2009)

Wikipedia. MS-DOS. 5 August 2009. http://en.wikipedia.org/wiki/MS-DOS (20 August 2009)

Wikipedia. Tcl. 13 August 2009. http://en.wikipedia.org/wiki/Tcl (20 August 2009)