Training Wheels for the Command Line

David Herrera

University of Texas at El Paso
El Paso Texas, 79968-0518
dh@alumni.utep.edu

Nigel Ward

University of Texas at El Paso
El Paso Texas, 79968-0518
nigel@cs.utep.edu

Abstract

Using the command line interface can be intimidating for novice users. Primarily designed for seasoned users, others who could potentially benefit from using it get stuck in its awkward and unfamiliar interface. The addition of "training wheels" has the potential to help such learners begin to "ride" the command line without "falling off" and getting discouraged. This paper proposes a "training wheels" interface, designed to support novice users in the step-by-step construction of executable commands, and so ease their transition to becoming a seasoned user of the command line. We created a "Command Line Training Wheels" system where the command line is augmented with various GUI features to support novice users. These features helped the user explore the functionality available; reduced memory load by providing visible, clickable options; provided task-relevant documentation at appropriate times; provided feedback during the composition of commands; and prevented some common errors. 22 lower-division computer science students were given a number of simple Unix tasks to complete using this system or using a Unix book and the naked command line. Most of the subjects preferred using the system and they tended to learn more when doing so.

1 Introduction

Power users use the command line interface (CLI), novices use graphical user interfaces (GUIs), and the threshold to progress from one to the other can be intimidating. The command-line interface, though a very powerful environment for communicating instructions to an operating system, requires not only previous knowledge of commands' syntax but also of constructs and special characters. Users are limited to keying in symbolic expressions and navigating through the system using almost exclusively the keyboard. Executing an instruction and then evaluating its result becomes a tremendous undertaking. Today even many otherwise technically-savvy users, including many computer science students, have never passed that hurdle, and suffer from awkward and slow systems control and information access.

One of the difficulties faced is in dealing with syntax errors. Novice users cannot be expected to construct syntactically correct expressions from memory nor recall enough detail to use options or special characters accordingly. Novice users can encounters errors hard to recover from or with cryptic messages. The CLI has many features that help, but these require practice and commitment.

We hypothesized that adding "training wheels"--- essentially wrapping a GUI around the command line ---would help people transition more easily into competent and confident command-line users. The GUI has the advantage of requiring less training to use proficiently. It makes available operations directly evident by listing them in menus. Some drawbacks are that it can complicate actions that are nested deeply in the menu hierarchy. This gets aggravated when an action must be repeated or is used often. Support for task automation, while helpful, is not widely available and can be difficult to use. These drawbacks are overcome with a CLI, but this is a solution only if novice users can learn to use it.

This paper examines how fusing the advantages provided by the GUI can help alleviate such an effort and encourage novice users to learn the CLI.

to appear in the proceedings of the 11th International Conference on Human-Computer Interaction (HCI International), to be published by Lawrence Erlbaum Associates, 2005.

2 Related Work

The idea of extending or improving the command line is of course not a new one. Past proposals include a GUI as a facade hiding the command line (Tyler, 1988) (Tyler & Treu, 1989), an expert system to critique or correct command line input (Jerrams-Smith, 1989), tools to visualize processing in pipes (Borg, 1990), and menu access to commonly used command line functionality (GNOME 2004). However there appears to have been no previous attempt to directly address the goal of easing the transition to the command line for GUI users.

Our idea of supplementing the command line with GUI functionality is directly inspired by Integrated Development Environments that provide GUI support for users working to construct a program. The primary difference here is command-line expressions tend to be conceptually and syntactically denser than program source code. We were also inspired by work taxonomizing the on-line information needs of users (Roesler, 1995). In terms of supporting learners, our work can be seen as a generalization of earlier "training wheels" research (Carroll & Carrithers, 1984); however we go beyond the previous focus --- making troublesome error states unreachable --- in an attempt to more closely approach the ideal of an exploratory environment supporting learning by doing. Another inspiration was the idea of providing "scaffolding" to enable learners to "engage in activities that would otherwise be beyond their abilities" (Soloway, 1996).

3 Features Needed in Training Wheels

Novice users are mostly used to interacting with the computer using the mouse. This allows direct manipulation for making selections, expressing actions, and exploring choices. The feedback provided for an action is immediate and the user can usually determine the effects as a result. The training required to use a GUI is far less than that to use the CLI. This suggests that an interface that supports novice users should provide support for mouse actions, give immediate feedback and permit the user to explore its features easily.

Another important feature needed is an easily accessible user-support system. Since many questions or unknowns arise when attempting to construct a command, having the necessary information at hand can help the user bridge the gulf between intention and execution and not get frustrated by the difficulty of finding help. A neatly arranged help system designed so that the user is able to access assistance only when needed or when appropriate can also prevent the user from being inundated by information. It allows for a command's options and arguments to be presented in smaller, more digestible chunks.

Finally, the system can help prevent common errors or errors that are difficult to recover from by catching them and providing the necessary guidance on how to correct the command. In particular, this shields new users from cryptic shell messages that may not make sense to them yet.

4 Design

Our earliest design ideas drew on our experiences as teachers and tutors; in particular situations where the teacher and the learner sit down with a sheet of paper, the learner attempts to compose a command, and the teacher gives explanations and rich feedback by annotating the shared representation on paper. The next step was generation of a paper prototype. At this stage we lost some of the richness of the interaction, as there was no easy way to provide highlighting, text decorations, and other graphical elements overlapping the command line textbox itself. However this was accepted as reasonable given the limitations of what is possible with standard widgets. The usability of this prototype was evaluated informally, and seemed to be appropriate for scenarios like the following:

> Joe User needs to find some information in some files. He knows he needs to specify a complex search, so he turns to Command Line Training Wheels. After clicking on "grep", he checks the description and knows that he's found the right command. The tool is clearly prompting him to enter something after the word "grep", so he knows that he needs to compose a complex command. Thanks to the syntax template displayed he knows he has three tasks: specifying what to look for, where, and how. Deciding to tackle the "where" first, he selects it and starts browsing the information that appears regarding file specification. Skimming the descriptions of the special characters, he sees that he needs to use a wildcard. Double-clicking it, * appears in the command line textbox, and he knows he's on his way to

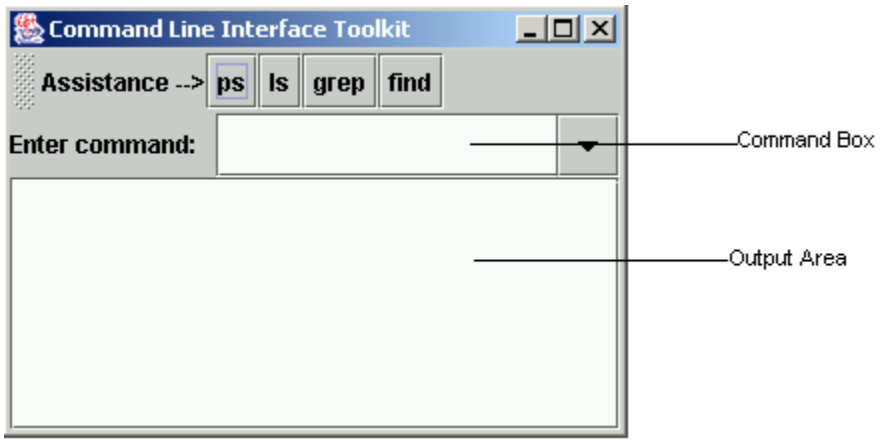

Figure 1: CLTW Gateway Window

completing the command. Wondering what will happen, he types return in the command line, to get a message advising him that a pattern is needed. He then proceeds to browse the information about patterns and clicks his way to creation of a full command. After executing he sees that the output was not what he wanted, so he returns to the command builder pane and uses it to revise his command. As he iteratively converges on a solution, his need for the GUI features gradually decreases and he finds himself more and more typing directly in the command line.

We then proceeded to final design and implementation. At this stage we lost the idea of a flexible, dynamic layout of information; for ease of implementation we switched to a more static, conventional organization of vertically separated panes of uniform width.

5 Implementation

We built a system, Command Line Training Wheels (CLTW), to test the hypothesis that using these techniques can support learning a command language. CLTW includes a generic window that mostly serves as gateway to the specific windows for each supported command, as seen in Figure 1. Command windows provide brief command descriptions, syntax templates, and other information; in a sense they are a restructuring of the Unix *man* pages. Figure 2 illustrates for the *ls* command. A command structure tree appears in the navigation window; this allows the user to work on a command one component at a time. Clicking on an item here brings up information specific to that component. Figure 3 illustrates by showing the information provided regarding regular expressions in grep.

6 Feature Summary

Command Line Training Wheels (CLTW) has 4 kinds of features, all applying well-known advantages of direct manipulation.

- Express action by making selections.

Pure command line interfaces require the user to remember many commands and options, as well as the syntactic structures of commands. CLTW provides alternative ways to assemble commands. For example, it is possible to click on examples to copy them to the command line, to click on checkboxes or radio buttons to add command options, to click on sets of characters to build up regular expressions, and to click on files and directories to add pathnames. In each case the item selected is inserted in the appropriate place in the command line.

- Feedback on actions.

Not only do GUI selections automatically update the command-line text box, the converse is also true. That is, when the user directly types a command in the text box, the appropriate checkboxes and choices (for example in the bottom pane of Figure 2) are updated to give the user feedback on the meaning of the command he is composing.

- Accessible information.

CLTW provides diverse help, examples, and reference information. This is presented in small, digestible pieces,

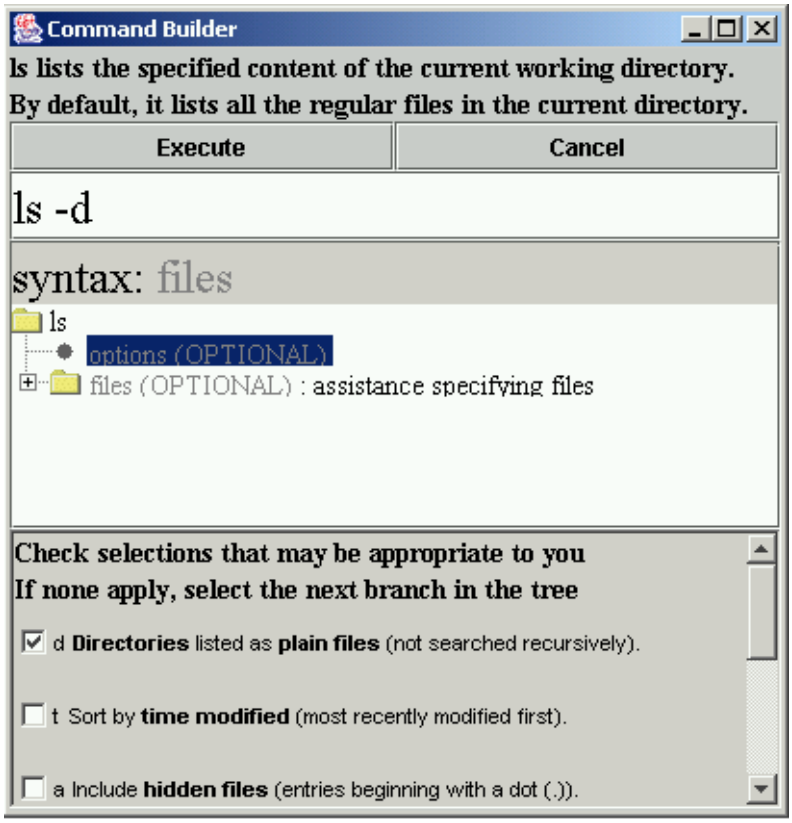

Figure 2: CLTW Command Window for *ls*

visible when appropriate and only when appropriate, and easily accessible. The navigation tree shown in Figure 2 shows how this information can be presented using the command's syntax structure. Clicking on one of these branches in the navigation tree displays information specific to that branch. Furthermore, as far as possible, information given by text is complemented by visual techniques to organize and provide information. For example, in Figure 3, text used in the syntax template is presented in two different colors. In this case, the gray font is used for *options* to suggest that they are optional while the red font is used for *files* and *patterns* suggesting they are required.

- Error-catching

If the user attempts to execute an incorrect or incomplete command, CLTW will provide an error message explaining what is needed. (This is done by checking command syntax and trapping most execution errors.) This not only informs novice users of the command usage, but also prevents them from falling into errors that can be hard to recover from.

Although these features are, we believe, valuable as training wheels for any command-line interface, they were perhaps especially valuable for Unix. This was because of several properties of Unix, including the fact that documentation of regular expressions is separate from documentation of the commands that use them, because of the formal structuring of the documentation (the man pages), because of the cryptically concise command names and options, because of feedback which is often absent or confusing, and because of the fact that commands which read from the standard input can lead to confusing failure modes.

7 Experiment and Results

CLTW was designed to help with hands-on learning of the Unix command line, by supporting the active participation of the user in the construction of commands. To determine if these techniques indeed can help novice users, we had subjects perform typical mid-level Unix tasks, such as searching for the number of lines containing a

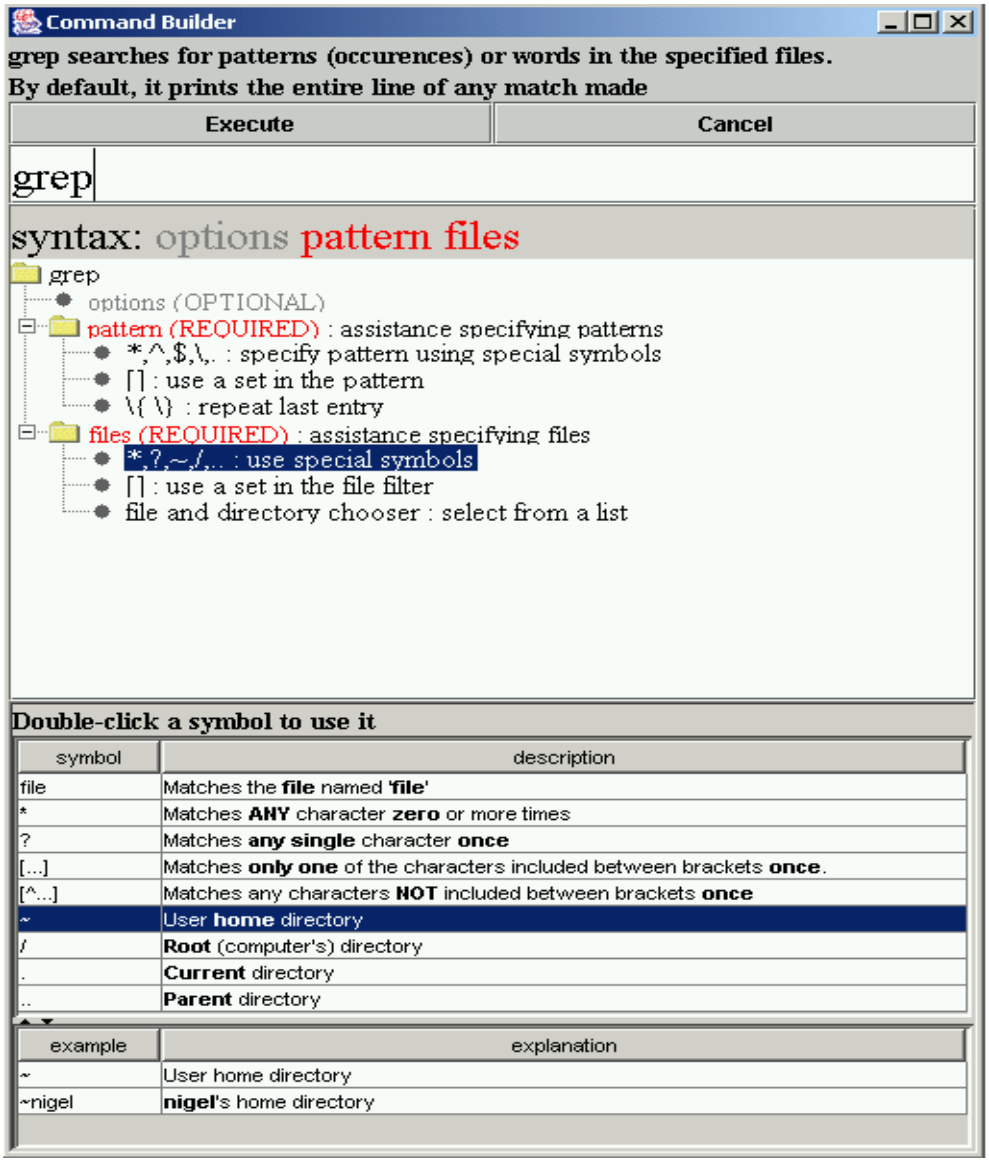

Figure 3: CLTW Window for *grep* including navigation information and regular expression information

specified substring in a certain collection of files. For comparison, in the same session, we had each subject also attempt a second set of tasks, of comparable difficulty, using the naked command line and a Unix book rich in examples (McMullen, 1998). The learning aid (book or CLTW) was varied so that one group used one aid with the first set of tasks and the second aid with the second set of tasks and vice-versa. We recruited subjects by posting flyers on bulletin boards in the Computer Science Building, offering a chance to experience Unix, advance science, and earn $10. 22 students volunteered, mostly lower division computer science students with little or no command line experience. Sessions took typically one to two hours. We intermittently observed subjects using the system, and their learning was evaluated with pre- and post-quizzes. Finally a questionnaire helped measure their own perceptions of their learning and the strengths and weaknesses of the system.

The number of questions answered correctly in the quiz and the number of tasks accomplished with correct commands were tabulated according to what aid was used. Statistically none of the findings was significant. However, it seems that subjects scored better on quizzes after using CLTW, although the same cannot be said for the number of commands successfully constructed. Figure 4 shows the total number of correct answers given by the

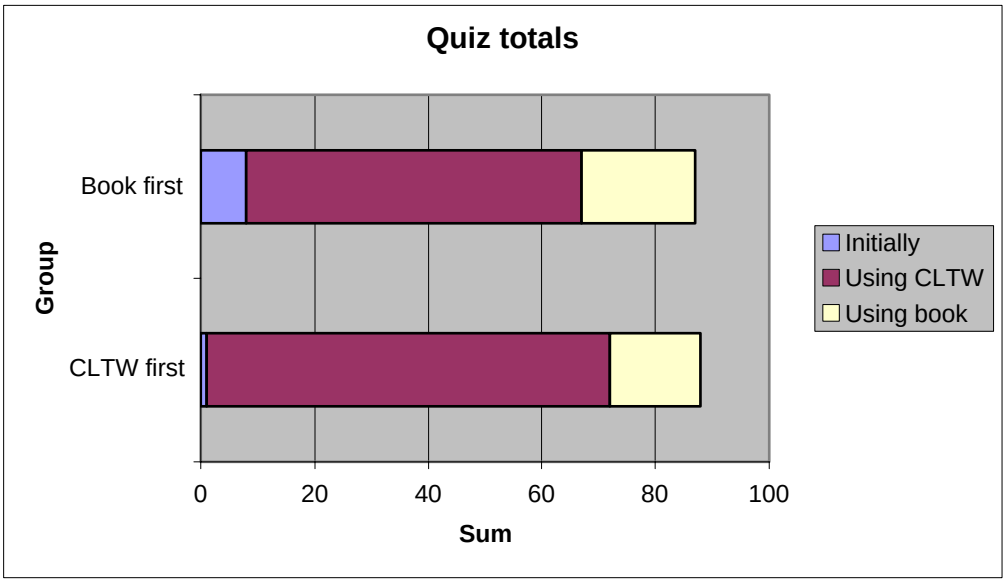

Figure 4: Unix proficiency initially and after each of two learning experiences

Table 1: Improvements in Unix Proficiency

Subject Group	Improvement after 1st set of tasks	Improvement after 2nd set of tasks	Total Improvement
Book first	59	20	79
CLTW first	71	16	87

group each time the quiz was taken. Table 1 shows the improvement in correct answers given after each set of tasks; for both sets, the group using CLTW learned more.

8 Discussion

Using our system, most subjects found the idea of the command line very interesting; many spent much more time on it than required. Most of the students liked the system overall: about half considered it better than the reference book. Of the rest, a few considered it equal in value to the book, a few thought both the book and the system were needed, and a couple thought the book alone was sufficient. Comments indicated that CLTW was perceived as more comfortable, more fun, faster, easier, providing more guidance, less error-prone and requiring less effort than directly composing commands with reference only to the book.

We were initially concerned that the "training wheels" could become a crutch that users would become dependent on, but most subjects tended to directly use the command line more over time.

On the minus side, subjects also noted weaknesses including bugs and limitations of the implementation and poorly worded help. Also, since the system was organized around Unix commands, rather than tasks, the system failed to help users figure out which command to use for each problem. Most seriously, it seemed that the system was less useful than the book for learning complex underlying concepts, such as regular expressions.

Perhaps the implementation was not as effective as it could be. Subjects could not always figure out how to use some of the techniques provided or the structure of the commands. Their responses suggested that documentation needed to be clearer and more descriptive. For example, while the group using CLTW seemed to correctly construct regular expressions, the command used was not always the appropriate one; subjects confused the difference

between ls, find and, in some cases, even grep. Error messages were not as helpful as they could be. They would simply specify that a command was incomplete, but did not specify exactly what part was missing.

There were several methodological issues that should be addressed in any follow-on experiment. First, the exact same information should have been presented in both aids. Second, the order of the tasks should have been rotated. This would eliminate any uneven distribution created by sets of unequal difficulty. Additionally, the types of learning measured by the quiz questions were eclectic: some questions asked for procedural knowledge while others conceptual knowledge. Finally, time measurement should have been done to determine how long a subject took per task or with each aid.

9 Summary

Overall the results suggest that GUI training wheels, wrapped around and tightly integrated with the command line, can help users attain proficiency with command languages. Subjects felt a high degree of comfort using CLTW to work with Unix. They felt that the syntax was understandable and that they could construct a command faster and more easily than using the book. One student liked not having to type in the commands, since typos could be made. They felt it provided more examples and guidance, which allowed them to locate pertinent information with less effort. One student said the software made learning better than the book, while another said repeated use of the software would help him learn more. Some common complaints included wanting more descriptive explanations and wanting more specificity for instructions pertaining to features of CLTW. The majority of user opinion suggests that CLTW was preferred to the book. Throughout the subjects' responses, the direct-manipulation facilities seem to be the driving force for this preference. One student described it as visual and hands-on. The results suggest that, with the aid of CLTW, users not proficient using command languages have a more familiar environment that facilitates their interaction with the computer through the CLI. Thus it seems that such techniques can help users to learn and use the Unix command language with less effort and more fun.

10 References

Borg, K. (1990). IShell: A Visual Unix Shell. In *CHI 1990*, 201-207.

Carroll, J.M., & Carrithers, C. (1984). Training Wheels in a User Interface. *Communications of the ACM*, 27, 800-806,

Gnome 2.2 desktop and developer platform. (2004). The GNOME Project. Retrieved June 3, 2004, from http://www.gnome.org/start/2.2/

Jerrams-Smith, J. (1989). An attempt to incorporate expertise about users into an intelligent interface for UNIX. *International Journal of Man-Machine Studies* 31, (3), 269–292.

McMullen, J. (1998). *UNIX Users Interactive Workbook (1st ed.).* Prentice Hall PTR,

Roesler, A.W., & McLellan, S.G. (1995). What Help do Users Need?: Taxonomies for On-line Information Needs and Access Methods. *CHI 1995*, 437-441.

Soloway, E., Jackson, S.L., et al. (1996). Learning Theory in Practice: Case Studies of Learner-Centered Design. *CHI 1996*, 189-196.

Tyler, S.W. (1988). Sauci: a knowledge-based interface architecture. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM Press, 235–240.

Tyler, S.W., & Treu S. (1989). An Interface Architecture to Provide Adaptive Task-Specific Context for the User. *International Journal of Man-Machine Studies* 30 (3), 303–327.