Usability improvements for products that mandate use of command-line interface: Best Practices

Samrat Dutta

M.Tech, International Institute of Information Technology, Electronics City, Bangalore
Software Engineer, IBM Storage Labs, Pune
samrat.dutta@iiitb.org

ABSTRACT

This paper provides few methods to improve the usability of products which mandate the use of command-line interface. At present many products make command-line interfaces compulsory for performing some operations. In such environments, usability of the product becomes the link that binds the users with the product. This paper provides few mechanisms like consolidated hierarchical help structure for the complete product, auto-complete command-line features, intelligent command suggestions. These can be formalized as a pattern and can be used by software companies to embed into their product's command-line interfaces, to simplify its usability and provide a better experience for users so that they can adapt with the product much faster.

INTRODUCTION

Products that are designed around a command-line interface (CLI), often strive for usability issues. A blank prompt with a cursor blinking, waiting for input, does not provide much information about the functions and possibilities available. With no click-able option and hover over facility to view snippets, some users feel lost. All inputs being commands, to learn and gain expertise of all of them takes time. Considering that learning a single letter for each command (often the first letter of the command is used instead of the complete command to reduce stress) is not that difficult, but all this seems useless when the command itself is not known. Then comes case sensitivity and combining it with the fact that almost all the products and applications that use command-line interfaces have their own syntax, is a lot to ask for. While this paper does not deal with the flaws of a CLI, these points have to be considered when we see a growing number of products using the command-line as their interface.

Let us consider a product (say, **Stornas**, a network attached storage (NAS) appliance) and map the above problems of CLI on that product. Now suppose Stornas mainly works on CLI for most of its operations. Being a NAS storage appliance, it will offer much functionality like exports, filesets, filesystems, replication and authentication, to name a few. And each of this functionality will consist of many commands. Even with a big product like Stornas, the command interface is still a blank screen. The user does not know about the product and its potential just by looking at the screen. In such cases, a complete mapping of the product (features, commands, components, etc.) and an intelligent CLI structure with loaded features in itself can boost the product.

Many system administrators (say **Sam**, who is the system administrator for Stornas) are ardent command-line users. Sam prefers the blank screen over graphical user interface (GUI) and can type commands with keyboards much faster than a regular user can do it by clicking on an option in the GUI. But even when executing several commands or a complete task, sometimes Sam prefers a GUI wrapper to execute such commands or uses an intelligent third-party shell tool to perform such tasks. This shows a flaw in Stornas, although an excellent NAS appliance, but since its command-line does not offer sound usability features, Sam has to use or at times be dependent on other tools and applications. This induces a need for Stornas (or any product and application that is solely dependent on the CLI) to provide much feature-rich capabilities and introduce many relevant and general usability features to enhance the users' experience and ease their tasks.

Consider, if Stornas had a modularized and functionality specific CLI help structure that organizes the complete product's surfing and usage along with high-quality command execution mechanisms, would Sam require any other tool? If Stornas is designed in such a way that the system itself guides Sam what components are present and what should be done next, then Sam can solely depend on Stornas and do not need to use any shell or GUI wrapper. This will make Stornas (or any product) more powerful and usable with clarity and simplicity. This paper intends to bring out some of the potential extensions to the existing command-line structures delivered by many products. Organizations and software firms can utilize these features in their products which are designed around CLI by engineering their command-line interfaces and subsequent help structures with ease.

CONSOLIDATED HIERARCHICAL CLI HELP FOR COMPLETE PRODUCT

Products or applications which are highly dependent on command-line rather than GUI, implicitly strive for usability excellence. But since the visualization of the CLI is just a blank screen and everything on it is text-based, forces it to achieve efficiency by providing added features. But GUIs can do anything with better usability than CLIs can but the vice versa is not true. Today most of the command-line interfaces are trying to embed features and functionality that are available in the GUI. One such feature that can be attached with products having mandate command-line interfaces is a complete hierarchical help structure for all the commands. While most similar products use online manual or documentation or command specific manual pages to cater to these help structures for all the commands, but still there is a lack of standard which enforces the use of the product CLI itself to build a complete help and reference sections within the command-line.

Linux contains a complete section-wise breakup of CLI description and man pages for each component, like user commands, system call, etc. which are then again broken up into sub-sections for each component and so on. This makes it very easy for the administrators (even a layman) to quickly get a feel of the command structure and perform operations fast and effectively. Considering that every product has a different CLI structure and syntax, a product specific extension, similar to what Linux provides, will give a much simpler way to browse around the components, commands, functionality, features and syntax of the product. The command-line should have a single place to search for everything about the product features, components, functionality, descriptions, commands and limitations. While 'man' commands does

a similar task for each command, but it goes out of picture when the product feature-related command is itself unknown to the user.

For example, if we consider our product Stornas, there can be several components like exports, users, filesystems, filesets, volume groups, disks, arrays, clusters, etc. There can be much functionality like authentication, authorization, mirroring, thin-provisioning, migration, replication, etc. If Sam, who is new to the product, is using this product and has to run some command for say, creation of an export on a fileset for a particular user, then Sam will have to browse through a lot of product manuals and help documents, a tedious job. Instead of this, if Stornas had a command-line help, structured in a complete layering format which Sam can browse while performing any operation, that can be an added usability efficiency.

UNIX has a consolidated CLI component-wise help, like [1]

(1) User Commands
(2) System Calls
(3) Library functions
(4) Devices
(5) File formats
(6) Games and Amusements
(7) Conventions and Miscellany
(8) System Administration and Privileged Commands
(L) Local. Some programs install their man pages into this section instead of (1)

Similarly, Stornas can also introduce section-wise break down of components, features which then extend to deeper levels as browsed, i.e. a single place to search everything about the product features, limitations, commands, etc.

Example [2]

```
(1) Exports
(2) Filesystems
(3) Filesets
(4) Event Logs
(5) Authentication
# man <Stornas> 1 complete
```

will list the Exports man page (NOT command-specific man page, but complete Export related man page including export related commands, limitations, features, etc.). Once Sam gets a feel of the complete Export functionality, he can then extend with export specific commands, or anything he wants. Once Sam comes out of the manual, he has the provision to then check specific commands or limitations like this,

```
# man <Stornas> 1 limitations // to check only the limitations
# man <Stornas> 1 commands // to check only the specific commands for export feature
```

Or Sam can also directly run a command man page to check the details of that command

specifically,
`# man <stornas_create_export>` // shows the man page of stornas_create_export

This kind of a complete and consolidated command-line help and manual structure, as a part of the product, will increase the efficiency of the users. It will also make the product complete in itself, removing the dependency on product manuals.

AUTOCOMPLETE CLI

In most of the existing command-line tools there is a feature of reverse lookup (using Ctrl+R) for commands. But this feature works on commands that are already typed by the user previously, that is, the reverse lookup searches for the commands that have already been executed. The method introduced here, provides a mechanism by which all the possible commands can be stored at a place and whenever a user starts to type any command, it will do a lookup from that stored list and automatically show the matches. The following example illustrates a similar scenario.

Example
Again returning to our product *Stornas*, consider there are five commands for Stornas. The entire commands' synopsis are stored in a file with possible options, as follows

```
stornas_create_export -export_name <value> -dependent_filesystem
<value> -export_owner <value>
stornas_list_files -parent_directory <value> -filename_filter
<value>
stornas_create_directory -parent_directory <value>
-directory_owner <value> -directory_size <value>
stornas_remove_export -export_name <value> -directory_name
<value>
stornas_delete_files -directory_name <value> -filename <value>
-size <value> -file_path <value>
```

Now, suppose our administrator, Sam, starts to type a command, it will dynamically preform a real-time lookup of all the commands, that start with it, from the file and show the matches.

```
# stornas_create_export -export_name <value>
-dependent_filesystem <value> -export_owner <value>
```

That is, as and when Sam begins to type a command (starting from the first character), a lookup operation on the stored commands will run in background that will check for the matches. So when Sam enters `stornas_r` (as shown in the following command), it will automatically do the background lookup search for all the commands that start with `stornas_r` and show the nearest resulting match. The example below illustrates this point (`stornas_r` is typed by Sam and the rest of the command is prompted in real-time by Stornas after doing a background lookup of all the commands that start with `stornas_r`)

```
# stornas_remove_export -export_name <value> -directory_name <value>
```

This feature, if embedded in products like Stornas which has compulsory command-line operations, then it will tremendously reduce the need for manual documentation for commands. Administrators like Sam can concentrate more on tasks and operations rather than searching around for command syntax and synopsis.

INTELLIGENT CLI SUGGESTIONS

Almost all enterprise products contain huge functionality and when the product is dependent on command-line instructions for performing operations, there are countless commands possible. Each functionality can be achieved by carrying out a series of steps, each with a separate command. So when one command is executed, often the next command remains the same, for achieving that particular functionality. Let us consider a similar use case in Linux:
Consider a case where Sam needs to perform the following tasks – Create a directory, create a file inside the directory, write something in that file, list the file and its contents and delete the file.
To perform this scenario, one possible way is to execute the following commands:

```
# mkdir /tmp/dir1
# ls /tmp/
dir1
# echo aaaa > /tmp/dir1/file1
# cat /tmp/dir1/file1
aaaa
# ls /tmp/dir1 | grep file1
file1
# rm /tmp/dir1/file1
```

So, the series of commands required every time to perform this scenario is: `mkdir, echo, cat, ls, rm`. The existing mechanism followed by most of the administrators is to write a script containing all the commands and then execute the script every time whenever the scenario needs to be performed. The script they write is often not a part of the product. Also different products have different command-line structures and different syntax. At such times, the administrators have to go through the product documentation for finding the commands related to a certain feature. This is where an intelligent command suggestion mechanism can be really useful and often reduce the detour to manual pages and product documentation, hence saving a lot of time.

Intelligent CLI suggestion is a method by which, when a command is typed, along with the results, the product will also show a list of suggested next commands, based on the output of the previous command. The following example will describe the above scenario with the intelligent command suggestion:

```
# mkdir /tmp/dir1
```

```
  Directory created successfully.
  Also look at – ls, echo, cat, rm
# ls /tmp/
  <lists all the objects inside the directory /tmp/>
  Also look at – echo, cat, rm, mkdir
```

The suggested commands after executing a command can be the same commands as shown in the manual pages for that command. The method described here is just another way to improve the usability of a product. Whenever there is a broad scenario, including pre-requisites, limitations, steps and assertions, the user can concentrate on the scenario, instead of searching around for next steps.

EXTENDING FEATURES AVAILABLE FOR COMMANDS TO COMMAND OPTIONS

Many products having command-line interface as their backbone for performing operations often depend on the underlying operating system for several features. Such features can be extended as part of the product itself. Two such extensions are shown below:

- As is mostly the case with many products based on CLI, like our Stornas, a <TAB> is often used to get the complete command that is half typed. If suppose, a user (say Sam, again) forgets a command, then presses the <TAB> to get the full command,

  Example
  `# stor` // presses <TAB>
  `stornas_create_export  stornas_list_files`
  `stornas_create_directory  stornas_remove_export`
  `stornas_delete_files`
  <TAB> shows the possible commands that are available.

  But often, many products like Stornas, limit features like <TAB>, grep, cut and other generic unix commands to only root users, since that can introduce security loop holes. Also these features are mostly not present for command specific options. In huge products, like Stornas, a command can have many options. There is often a possibility that the user might not understand what option is applicable for a particular operation. For example, suppose the command `stornas_create_export` has the following options: `export_name, dependent_filesystem, export_owner`
  Then a way to search for the command options can be introduced by the product itself, something like

  `# stornas_create_export -export_name <value> -` // presses <TAB> or any other mechanism suitable for the product,
  `-dependent_filesystem`
  `-export_owner`
  // showing the options of the command `stornas_create_export`

will be very useful. The user will not have to remember command options or search for manual documentation for options. This method will increase the command execution rate drastically.

- Similarly, as above, there can be another product specific extension to the existing feature which most of the products follow. Users often use the `-help` option for finding a quick help or synopsis of a command.

  Example
  `# stornas_create_directory -help` // shows the basic help of `stornas_create_directory` command

  But often, products do not have a quick help for command-specific options. For example, a command like this will not work on many products:
  `stornas_create_directory -directory_name -help` // trying to get the help of the `-directory_name` option
  This method can be introduced as a feature for any product. It will improve the readability of the options and make the users aware of command specific options on the go.

If something like these can also be included or added as a product feature, it can help the users immensely. A command option in itself might be very complex. There can be several scenarios in which the same command can be executed. It will help the users to not get stuck for thinking or searching for an option. These kind of features are more applicable on products that do not link their usage with existing unix-like CLI offerings.

CONCLUSION

Products or applications projected on a mandatory command-line structure, will have an added boost if these kinds of qualities are provided by the product itself, Users of the product will love it and will try to stick to it. The command-line interface is the one that is visible to the user in these products. Everything revolves around the command-line interface. It should always be considered how much productive and plentiful the command-line is. While there can be many such improvements, the ones described in this paper can be used across all the products built on an underlying command-line interface. This will give a modular and systematic approach for users with extreme clarity and simplicity, so that no user returns after using the product once. Often customers get attracted to products with excellent interfaces. These features will reduce the efforts of administrators and will be major usability advancements.

REFERENCES

[1] Norman Robinson, *Reading Man Pages*, [Online]. Available: http://www.linuxcommand.org/reading_man_pages.php

[2] Mary Lovelace, Vincent Boucher, Shradha Nayak, Curtis Neal, Lukasz Razmuk, John Sing, John Tarella, *IBM Redbooks*, *IBM Scale Out Network Attached Storage (SONAS) Architecture, Planning, and*

Implementation Basics. Available: http://www.redbooks.ibm.com/

[3] Max Steenbergen, *Command Line: Alive & Kicking,* UX Maganize, Article No. 575 November 4, 2010. [Online]. Available: http://uxmag.com/articles/command-lines-alive-kicking

[4] Ramesh Natarajan, *LINUX 101 HACKS, Practical Examples to Build a Strong Foundation in Linux,* Ch. 8: Command Line History. Available: http://thegeekstuff.com

[5] Don Norman, *UI Breakthrough-Command Line Interfaces,* Column written for *Interactions, volume 14, issue 3*. © CACM, 2007. Available: http://www.jnd.org/dn.mss/ui_breakthrough-comm.html

[6] Scott Rippee, *Getting Started with BASH – A Bash Tutorial*, Available: http://www.hypexr.org/bash_tutorial.php

[7] Ian Macdonald, Working more productively with bash 2.x/3.x. Available: http://www.caliban.org/bash/

[8] Lee Holmes, *Windows PowerShell Cookbook, 3rd Edition (*2013*)*, O'Reilly Media, Inc., 978-1-4493-2068-3, Ch. 19.: Integrated Scripting Environment