Bachelor Thesis
The Nut Shell – A Framework for Creating Interactive Command Line Tutorials

Sebastian Morr
2013–11–27
Motivation

- *Command line*: Powerful, efficient user interface
Motivation

- **Command line**: Powerful, efficient user interface
- **But**: Steep learning curve
Motivation

- **Command line**: Powerful, efficient user interface
- **But**: Steep learning curve
- **Common teaching approach**: Static text. Inflexible!
Motivation

- **Command line**: Powerful, efficient user interface
- But: Steep learning curve
- Common teaching approach: Static text. Inflexible!
- This thesis: More direct, interactive teaching approach
Motivation

- *Command line*: Powerful, efficient user interface
- But: Steep learning curve
- Common teaching approach: Static text. Inflexible!
- This thesis: More direct, interactive teaching approach

Core idea

Interweave tutorial text with CLI output and react to user’s commands, the system’s state and output.
Motivation

- **Command line**: Powerful, efficient user interface
- But: Steep learning curve
- Common teaching approach: Static text. Inflexible!
- This thesis: More direct, interactive teaching approach

Core idea

Interweave tutorial text with CLI output and react to user’s commands, the system’s state and output.

- Inspiration: Text adventures!
Example *Nut Shell* session

```
$ cd kitchen

$ ls

$ mv elephant fridge

[The elephant does not fit into the fridge]

Oh, it doesn’t seem to be that easy. Can you find out how big the file is? The man page of `ls` will help you!

$ ls

$ man ls

[Display of the man page, skipped here]
```
Example *Nut Shell* session

$ cd kitchen

[The elephant does not fit into the fridge]

Oh, it doesn't seem to be that easy. Can you find out how big the file is? The man page of `ls` will help you!

$ ls elephant fridge/

$ man ls

[Display of the man page, skipped here]
Example *Nut Shell* session

```
$ cd kitchen
$
```

[The elephant does not fit into the fridge]

Oh, it doesn't seem to be that easy. Can you find out how big the file is? The man page of `ls` will help you!

```
$ ls
$ ls
$ man ls
```

[Display of the man page, skipped here]
Example *Nut Shell* session

```bash
$ cd kitchen
$ ls
```

---

**Sebastian Morr | The Nut Shell | 3**

**Institut für Programmierung und Reaktive Systeme**

---

**Technische Universität Braunschweig**
Example *Nut Shell* session

```bash
$ cd kitchen
$ ls
elephant fridge/
$ 
```
Example *Nut Shell* session

```bash
$ cd kitchen
$ ls
elephant fridge/
$ mv elephant fridge
```
Example *Nut Shell* session

$ cd kitchen
$ ls
elephant fridge/
$ mv elephant fridge

[The elephant does not fit into the fridge]

Oh, it doesn’t seem to be that easy. Can you find out how big the file is? The man page of ls will help you!

$
Example *Nut Shell* session

```
$ cd kitchen
$ ls
elephant fridge/
$ mv elephant fridge

[The elephant does not fit into the fridge]

Oh, it doesn’t seem to be that easy. Can you find out how big the file is? The man page of ls will help you!

$ ls
```
Example *Nut Shell* session

```
$ cd kitchen
$ ls
elephant fridge/
$ mv elephant fridge

[The elephant does not fit into the fridge]

Oh, it doesn’t seem to be that easy. Can you find out how big the file is? The man page of ls will help you!

$ ls
elephant fridge/
$ 
```
Example *Nut Shell* session

$ cd kitchen
$ ls
elephant fridge/
$ mv elephant fridge

*The elephant does not fit into the fridge*

Oh, it doesn’t seem to be that easy. Can you find out how big the file is? The *man page of ls* will help you!

$ ls
elephant fridge/
$ man ls
Example *Nut Shell* session

$ cd kitchen
$ ls
elephant fridge/
$ mv elephant fridge

[The elephant does not fit into the fridge]

*Oh, it doesn’t seem to be that easy. Can you find out how big the file is? The man page of ls will help you!*

$ ls
elephant fridge/
$ man ls

[Display of the man page, skipped here]
Example *Nut Shell* session (cont.)

```bash
$ ls -l elephant
-rw------- 1 seb users 10485760 27. Okt 22:25 elephant
```

Okay, about ten million bytes. `ls` has the option `-sh` to display that in a more comprehensible order of magnitude.

```bash
$ ls -sh
10M elephant
```

10 megabytes? Indeed, the fridge isn’t that large. We have to make the elephant smaller.
Example *Nut Shell* session (cont.)

```
$ ls -l elephant
```

Okay, about ten million bytes. `ls` has the option `-sh` to display that in a more comprehensible order of magnitude.

```
$ ls -sh
10M elephant
```

10 megabytes? Indeed, the fridge isn't that large. We have to make the elephant smaller.
$ ls -l elephant
-rw------- 1 seb users 10485760 27. Okt 22:25 elephant

Okay, about ten million bytes. `ls` has the option `-sh` to display that in a more comprehensible order of magnitude.
Example *Nut Shell* session (cont.)

```shell
$ ls -l elephant
-rw------- 1 seb users 10485760 27. Okt 22:25 elephant

Okay, about ten million bytes. *ls* has the option `-sh` to display that in a more comprehensible order of magnitude.

$ ls -sh
```
Example *Nut Shell* session (cont.)

```
$ ls -l elephant
-rw------- 1 seb users 10485760 27. Okt 22:25 elephant

Okay, about ten million bytes. *ls* has the option `-sh` to display that in a more comprehensible order of magnitude.

$ ls -sh
10M elephant

10 megabytes? Indeed, the fridge isn’t that large. We have to make the elephant smaller.
Overview

Goal

Design, implementation, application and evaluation of a framework that allows the creation of command line tutorials with this interactive teaching approach: The *Nut Shell*. 
Overview

Goal

Design, implementation, application and evaluation of a framework that allows the creation of command line tutorials with this interactive teaching approach: The *Nut Shell*.

Outline
Overview

Goal

Design, implementation, application and evaluation of a framework that allows the creation of command line tutorials with this interactive teaching approach: The *Nut Shell*. 

Outline

1. Construct abstraction layer for uniform access to arbitrary CLIs
Overview

Goal

Design, implementation, application and evaluation of a framework that allows the creation of command line tutorials with this interactive teaching approach: The Nut Shell.

Outline

1. Construct abstraction layer for uniform access to arbitrary CLIs
2. Introduce new language to describe tutorial lessons
Overview

Goal

Design, implementation, application and evaluation of a framework that allows the creation of command line tutorials with this interactive teaching approach: The Nut Shell.

Outline

1. Construct abstraction layer for uniform access to arbitrary CLIs
2. Introduce new language to describe tutorial lessons
3. Comparative evaluation with about 120 participants
Outline

Introduction

The CLI Abstraction Layer

The nutsh Language

Application and Evaluation
Purpose

- Goal: Common interface to all supported CLIs
Purpose

- Goal: Common interface to all supported CLIs
- Recognize parts of the command line interaction:
Purpose

- Goal: Common interface to all supported CLIs
- Recognize parts of the command line interaction:
  1. Prompt
Purpose

- **Goal:** Common interface to all supported CLIs
- **Recognize parts of the command line interaction:**
  1. Prompt
  2. Command
Purpose

- **Goal:** Common interface to all supported CLIs
- **Recognize parts of the command line interaction:**
  1. Prompt
  2. Command
  3. Output
Purpose

- **Goal:** Common interface to all supported CLIs
- **Recognize parts of the command line interaction:**
  1. Prompt
  2. Command
  3. Output
- **Keep all editing features intact**
Purpose

- Goal: Common interface to all supported CLIs
- Recognize parts of the command line interaction:
  1. Prompt
  2. Command
  3. Output
- Keep all editing features intact
- Maintain the CLI’s state
Architecture

Keyboard  Screen
Architecture

- Keyboard
- Screen
- Terminal
Architecture

- Keyboard
- Terminal
- Screen
- Input filter
Architecture

1. Keyboard
2. Screen
3. Terminal
4. Input filter
5. CLI process
Architecture
Architecture

Keyboard <-> Screen

Terminal

Logic

Input filter <-> Tokenizer

tokens

CLI process
Architecture

- Keyboard
- Screen
- Terminal
- Logic
- Input filter
- Tokenizer
- CLI process
Architecture

- Keyboard
- Screen
- Terminal
- Logic
- Input filter
- Tokenizer
- CLI process

(tokens)
Requirements

- Abstraction layer has to rely on common features of CLIs:
Requirements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
Requiements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
Requirements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
     - Ctrl+E to jump to the end of the line
Requirements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
     - \text{Ctrl} + \text{E} to jump to the end of the line
     - \text{Ctrl} + \text{U} to delete current line, put it in a buffer
Requiements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
     - Ctrl+E to jump to the end of the line
     - Ctrl+U to delete current line, put it in a buffer
     - Ctrl+Y to reinsert the buffer
Requiements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
     - Ctrl + E to jump to the end of the line
     - Ctrl + U to delete current line, put it in a buffer
     - Ctrl + Y to reinsert the buffer
- Examples:
Requiements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
     - Ctrl + E to jump to the end of the line
     - Ctrl + U to delete current line, put it in a buffer
     - Ctrl + Y to reinsert the buffer

- Examples:
  - System shells: Bash, tcsh, zsh, ...
Requiements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
     - \texttt{\textbackslash Ctrl} + \texttt{E} to jump to the end of the line
     - \texttt{\textbackslash Ctrl} + \texttt{U} to delete current line, put it in a buffer
     - \texttt{\textbackslash Ctrl} + \texttt{Y} to reinsert the buffer

- Examples:
  - System shells: Bash, tcsh, zsh, . . .
  - REPL-loops of programming languages (Ruby, Python, Haskell, . . .)
Requirements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
     - \texttt{Ctrl} + \texttt{E} to jump to the end of the line
     - \texttt{Ctrl} + \texttt{U} to delete current line, put it in a buffer
     - \texttt{Ctrl} + \texttt{Y} to reinsert the buffer

- Examples:
  - System shells: Bash, tcsh, zsh, ...
  - REPL-loops of programming languages (Ruby, Python, Haskell, ...)
  - SQL consoles
Requiements

- Abstraction layer has to rely on common features of CLIs:
  1. User customizable prompts.
  2. Readline-style keybindings:
     - Ctrl+E to jump to the end of the line
     - Ctrl+U to delete current line, put it in a buffer
     - Ctrl+Y to reinsert the buffer
  
- Examples:
  - System shells: Bash, tcsh, zsh, ...
  - REPL-loops of programming languages (Ruby, Python, Haskell, ...)
  - SQL consoles
  - Mathematics software (Gnuplot, Sage, Octave)
Approach

- Use special *markers* for annotation
Approach

- Use special *markers* for annotation
  - Suitable choice: Unicode code points from the *Private Use Area* (U+E000 – U+F8FF)
Approach

- Use special *markers* for annotation
  - Suitable choice: Unicode code points from the *Private Use Area* (U+E000 – U+F8FF)
- Insert into prompt, do not display
Approach

- Use special *markers* for annotation
  - Suitable choice: Unicode code points from the *Private Use Area* (U+E000 – U+F8FF)
- Insert into prompt, do not display
- `inputFilter`: Wait for *line feed*, send sequence to repeat command between markers
Command line operations

- Abstraction layer generates token stream
Command line operations

- Abstraction layer generates token stream
- Two Operations:
Command line operations

- Abstraction layer generates token stream
- Two Operations:
  1. Prompt the user for a command
Command line operations

- Abstraction layer generates token stream
- Two Operations:
  1. Prompt the user for a command
     - Write Prompt token to the terminal
Command line operations

- Abstraction layer generates token stream
- Two Operations:
  1. Prompt the user for a command
     - Write Prompt token to the terminal
     - Store Command tokens as the user's command
Command line operations

- Abstraction layer generates token stream
- Two Operations:
  1. Prompt the user for a command
     - Write Prompt token to the terminal
     - Store Command tokens as the user’s command
     - Store Output token as the command’s output
Command line operations

- Abstraction layer generates token stream
- Two Operations:

1. Prompt the user for a command
   - Write Prompt token to the terminal
   - Store Command tokens as the user’s command
   - Store Output token as the command’s output

2. Send a hidden command to the CLI
Command line operations

- Abstraction layer generates token stream
- Two Operations:

1. Prompt the user for a command
   - Write Prompt token to the terminal
   - Store Command tokens as the user’s command
   - Store Output token as the command’s output

2. Send a hidden command to the CLI
   - Send command directly to *Input Filter*
Command line operations

- Abstraction layer generates token stream
- Two Operations:
  1. Prompt the user for a command
     - Write Prompt token to the terminal
     - Store Command tokens as the user’s command
     - Store Output token as the command’s output
  2. Send a hidden command to the CLI
     - Send command directly to *Input Filter*
     - Capture command and output tokens, but don’t display them
Outline

Introduction

The CLI Abstraction Layer

The nutsh Language

Application and Evaluation

Sebastian Morr | The Nut Shell | 12

Institut für Programmierung und Reaktive Systeme
Design goals

- As easy to read and write as possible
Design goals

- As easy to read and write as possible
  - Syntax resembles C and Go
Design goals

- As easy to read and write as possible
  - Syntax resembles C and Go
  - Use regular expressions
Design goals

- As easy to read and write as possible
  - Syntax resembles C and Go
  - Use regular expressions
- Keep language as small as possible, but powerful enough
Design goals

- As easy to read and write as possible
  - Syntax resembles C and Go
  - Use regular expressions
- Keep language as small as possible, but powerful enough
- Syntactic support for often-used semantical constellations
Lexical elements

As usual:

- Comments
- White space
- Identifiers
- Keywords: break, def, else, if, prompt, return
- Operators, delimiters
- String literals
String Expressions

- Concatenation: "foo" + "foo"
String Expressions

- Concatenation: "foo"+"foo"
- Check for equality: "foo"+"foo" == "foofoo"
String Expressions

- Concatenation: "foo"+"foo"
- Check for equality: "foo"+"foo" == "foofoo"
- Check for (exact) regex match: "foo" =~ "f[aio]."
String Expressions

- Concatenation: "foo" + "foo"
- Check for equality: "foo" + "foo" == "foofoo"
- Check for (exact) regex match: "foo" =~ "f[ai0]."
- Empty string has truth value false, others true
String Expressions

- Concatenation: "foo"+"foo"
- Check for equality: "foo"+"foo" == "foofoo"
- Check for (exact) regex match: "foo" =~ "f[aio]."
- Empty string has truth value false, others true
- Boolean operators: !, &&, and || as usual
Built-in functions

- say: Output explanation text (indented, colored)
  
say("This is explaining text.")

"This is the short form."
Built-in functions

- **say**: Output explanation text (indented, colored)

  `say("This is explaining text.")`

  "This is the short form."

- **run**: Execute hidden command, return output

  `run("1+1")`
If statements

```plaintext
if "test" == "test" {
    "Everything is OK."
} else {
    "Wait, what?"
}
```
Prompt statements

"Please calculate the product of 6 and 7."

```
prompt {
  if output == "42" {
    break
  } else {
    "Please try again."
  }
}
"Well done!"
```

Infinite loop, prompt user for command before each pass.
Define command and output functions
Function definitions

Only at top level, avoid name masking!

```python
def say_twice(text) {
    say(text)
    say(text)
}

say_twice("Hey!")
```
Nesting statements

Use case: Check same conditions for group of prompt statements.
Nesting statements

Use case: Check same conditions for group of prompt statements.

```python
def respond_to_help {
    if command =~ "help" {
        "Sorry, you’re on your own."
    }
}
```

```python
respond_to_help {
    prompt { /* ... */ }
    prompt { /* ... */ }
}
```
**Parsing**

- `nutsh` has $LR(1)$ grammar: Can be parsed by a bottom-up parser with lookahead 1 reading from left to right in a single pass, creating a rightmost derivation
 Parsing

- `nutsh` has $LR(1)$ grammar: Can be parsed by a bottom-up parser with lookahead 1 reading from left to right in a single pass, creating a rightmost derivation
- Framework uses a standard parser generator, YACC
 Parsing

- *nutsh* has *LR(1)* grammar: Can be parsed by a bottom-up parser with lookahead 1 reading from left to right in a single pass, creating a rightmost derivation
- Framework uses a standard parser generator, YACC
- Parser creates a *syntax tree*
Interpretation

- Function definition: Added to the symbol table (no scoping)
Interpretation

- Function definition: Added to the symbol table (no scoping)
- String expressions: Value can be *synthesized*
Interpretation

- Function definition: Added to the symbol table (no scoping)
- String expressions: Value can be *synthesized*
- Lazy evaluation, *pass-by-value*
Interpretation

- Function definition: Added to the symbol table (no scoping)
- String expressions: Value can be *synthesized*
- Lazy evaluation, *pass-by-value*
- Nesting statements: Calls are pushed on a stack when entering, and are removed when leaving the statement
Automated testing

- Goal: Automatic verification of lessons

```perl
expect run("text = 'stressed'")
"Reverse the content of `text` and save it in `text2`!"

prompt {
  if test("text2 == 'desserts'") {
    expect("text2 = text.reverse")
    expect("text.reverse!; text2 = text")
    break
  } else {
    expect("text2 = 'somethingdifferent'")
  }
}
```
Automated testing

- Goal: Automatic verification of lessons
- Provide built-in function `expect`

```bash
run("text = 'stressed'")
"Reverse the content of `text` and save it in `text2`!

prompt{
if test("text2 == 'desserts'") {
  expect("text2 = text.reverse")
  expect("text.reverse!; text2 = text")
  break
} else {
  expect("text2 = 'somethingdifferent'")
}
```
Automated testing

- Goal: Automatic verification of lessons
- Provide built-in function `expect`

```plaintext
run("text = 'stressed'")
"Reverse the content of `text` and save it in `text2`!"

prompt {
  if test("text2 == 'desserts'") {
    expect("text2 = text.reverse")
    expect("text.reverse!; text2 = text")
    break
  } else {
    expect("text2 = 'somethingdifferent'")
  }
}
```
Implementation

- Framework is implemented in Go: Concurrency with synchronized communication, big standard library with Unicode support
Implementation

- Framework is implemented in Go: Concurrency with synchronized communication, big standard library with Unicode support
- 2576 source lines of code
Implementation

- Framework is implemented in Go: Concurrency with synchronized communication, big standard library with Unicode support
- 2576 source lines of code
- Tutorial representation:
Implementation

- Framework is implemented in Go: Concurrency with synchronized communication, big standard library with Unicode support
- 2576 source lines of code
- Tutorial representation:
  - Directory, contains several lesson files written in *nutsh*
Outline

Introduction

The CLI Abstraction Layer

The nutsh Language

Application and Evaluation
Preparatory computer science courses at the Braunschweig University of Technology exists since 2003
Preparatory computer science courses at the Braunschweig University of Technology exists since 2003

In the fall semester 2013–2014: 150 students enrolled
Setting

- Preparatory computer science courses at the Braunschweig University of Technology exists since 2003
- In the fall semester 2013–2014: 150 students enrolled
- Split into two groups: Two thirds Nut Shell, one third paper exercises
2875 lines of *nutsh* code:
2875 lines of *nutsh* code:

1. Introduction - first examples with `cal`
2. Looking and moving around - `ls` and `cd`
3. Helping users to help themselves - `man`
4. File system and paths
5. Creating and editing files - `mkdir` and editors
6. History and tab completion
7. Java
8. Deleting files and directories - `rmdir`, `rm`
9. Copying, moving and linking files - `cp`, `mv`, `ln`
10. Process management - `ps`
11. Aliases
12. Variables
Content (cont.)

13. Commandline editing
14. Wildcards
15. Quoting
16. Compressing files - tar, gzip, bzip2
17. Redirection and pipes
18. Looking for patterns - grep
19. Small useful commands
20. wget and curl
21. Typesetting with \LaTeX
22. Java, part 2
23. Comparing files - cmp, diff, patch
24. Searching - find, locate
25. Sorting
26. Shell scripts
27. Version control with Git
28. Working remotely with SSH
29. Automation with makefiles
Style

- Basic teaching style:
Style

- Basic teaching style:
  - State a general problem
Style

- Basic teaching style:
  - State a general problem
  - Present method or tool for solving this class of problems using a simple example
Style

- **Basic teaching style:**
  - State a general problem
  - Present method or tool for solving this class of problems using a simple example
  - Pose problems of increasing difficulty
**Basic teaching style:**

- State a general problem
- Present method or tool for solving this class of problems using a simple example
- Pose problems of increasing difficulty

- Often multiple solutions
Style

- Basic teaching style:
  - State a general problem
  - Present method or tool for solving this class of problems using a simple example
  - Pose problems of increasing difficulty
- Often multiple solutions
- Let user choose among several paths
Style

- Basic teaching style:
  - State a general problem
  - Present method or tool for solving this class of problems using a simple example
  - Pose problems of increasing difficulty
- Often multiple solutions
- Let user choose among several paths
- Use analogies, virtual “home” environment
After the sixth day, online survey with three parts:

1. General statements, rated from 1 to 10 & help/day
Survey

After the sixth day, online survey with three parts:

1. General statements, rated from 1 to 10 & help/day
2. Test with 12 questions by neutral third
After the sixth day, online survey with three parts:

1. General statements, rated from 1 to 10 & help/day
2. Test with 12 questions by neutral third
3. Nut Shell assessment
Results

First part: 64 answers in total. 53 Nut Shell users, 11 exercise sheet users
Results

First part: 64 answers in total. 53 Nut Shell users, 11 exercise sheet users
Results (cont.)

Few participants in the control group. Warped results?
Results (cont.)

Help/day

Score
Results (cont.)

- Help/day
- Score
- Explanations
- Tips
- Exercises
- Use again

Few participants in the control group. Warped results?
Results (cont.)

Few participants in the control group. Warped results?
Results (cont.)

Day

Participants
Discussion

- High participant loss is representative
Discussion

- High participant loss is representative
- Main results:
Discussion

- High participant loss is representative
- Main results:
  - Nut Shell motivated students to attend to the course
Discussion

- High participant loss is representative
- Main results:
  - Nut Shell motivated students to attend to the course
    - Having over 63% attending over the whole timespan is highly gratifying!
Discussion

- High participant loss is representative
- Main results:
  - Nut Shell motivated students to attend to the course
    - Having over 63% attending over the whole timespan is highly gratifying!
  - Lowered demand for external help: More independent students
Due to positive effects, Nut Shell will be used for upcoming preparatory courses.
Conclusions

- Due to positive effects, Nut Shell will be used for upcoming preparatory courses
- Another institute has shown interest to use Nut Shell for a Git course
Conclusions

- Due to positive effects, Nut Shell will be used for upcoming preparatory courses
- Another institute has shown interest to use Nut Shell for a Git course
- Participant wants to use Nut Shell to teach command line concepts to pupils
Conclusions

- Due to positive effects, Nut Shell will be used for upcoming preparatory courses
- Another institute has shown interest to use Nut Shell for a Git course
- Participant wants to use Nut Shell to teach command line concepts to pupils
- Student’s general feedback very positive
Conclusions

- Due to positive effects, Nut Shell will be used for upcoming preparatory courses
- Another institute has shown interest to use Nut Shell for a Git course
- Participant wants to use Nut Shell to teach command line concepts to pupils
- Student’s general feedback very positive
- Software and tutorial will be released under a free, open source license
Summary

Content of the thesis

1. Design
   - Universal CLI abstraction layer
   - DSL for writing and testing lessons
2. Implementation
3. Application
   - Bash tutorial with 29 lessons
4. Evaluation
Thank you!

Questions?