# CLI Usability

### Tyler Cecil, Louis Jencka, Jesse Maes

### April 1, 2016

# 1    Background

In the lecture from the first day of Human Computer Interaction, we talked about the history of computer user interfaces. The command line interface (CLI) was mentioned in a slide, as an example of a historical type of interface. It was described as "efficient, precise, and fast."

Unfortunately, this advanced interface has gained its power at the expense of ease-of-use. Utilizing the CLI entails a large overhead to learning set of commands, which is what has led to peculiar design goals for command line interfaces. Commands and options are generally terse abbreviations, which are very easy to memorize, but are not necessarily easy to learn.

As Computer Scientists, the CLI isnt just relic of history; we make heavy use it on a daily basis. Its a powerful and highly flexible tool, useful for a menagerie of tasks. For the beginner user, the shell can be an incomprehensible nightmare – its blinking cursor against a blackened sky a stand-in for man's helpless struggle against death. Just as text editors have made leaps and bounds in communicating essential contextual information, we believe the CLI can be improved for beginning and advanced users alike.

# 2    Potential Users

## 2.1    Beginners

Our first imagined user is brand new to the shell, if not Linux itself. A freshman CSE student, for example, would need to learn how to navigate a UNIX operating system via its main interface - the shell. As a first-time Linux/BSD user they would need to learn some basic information about how their system is organized, and how to manipulate it. From their previous computer usage they could be expected to understand the hierarchical nature of file systems, but nothing more. What this user needs is to reach a baseline fluency in CLI usage: how to navigate their file system; how to create and modify files; how to learn more.

1. No experience with UNIX operating systems

2. No experience with a CLI

3. No experience with the state inherent to a shell, mainly the current working directory

4. No knowledge of what programming is, or how to automate tasks

5. No knowledge of the anatomy of a command (command name, arguments, options, et cetera)

6. No knowledge of the software available on their system

7. No knowledge of basic OS tasks, such as installing software

8. Has experience with common GUI techniques - click-to-open, drag-and-drop, et cetera

9. Is uncertain

## 2.2   Casual Users

Another potential user would be for scientists, engineers, or anyone with experience as a technical computer user whose workflow could be more efficient if they knew how to use the shell. The CLI can make it easier for users to automate tasks and make precise changes to the files they are working on. These users may already know how to complicated tasks on their computers, or even be able to write programs. Many powerful tools insulate their users from the details they would need to use the shell, such as Matlab, Maple, or Excel. An information-rich user interface could help casual power users make the transition to understand how to use their terminals for more complicated tasks such as shell scripting and task automation.

1. Understands the basics of how computers work and filesystems are organized

2. Understands what computer programs are or has knowledge of how to program

3. Is only familiar with a GUI, or is more comfortable with pointing and clicking than typing

4. Regularly needs to perform some repetitive task on their computer, but does not imagine the task could be automated

5. Finds themselves lost if/when they use the CLI - doesnt understand the larger context of the system surrounding the shell

6. Is unaware of most of the programs, commands, and command line options they could be using (has only seen the tip of the iceberg)

7. Does not know how to apply technical knowledge or programming skills to system tasks in the CLI

## 2.3 Advanced User

We also believe that advanced users who already know their way around the command line could benefit from an information-rich shell. This user can be expected to know by heart the commands they need to use, the details of how their system is organized, and how to program in a shell language. Where the CLI could stand improvement for these users is in providing more details on the state of their system, and allowing a terser search of console-oriented documentation.

1. Deep understanding of OS structure and interfaces

2. Mastery of command line usage

3. Commonly constructs aliases and automates tasks

4. Consults system documentation to find advanced command options

5. Understands that task inference is frequently achievable from context in the shell

# 3 Tasks

## 3.1 User Stories

*As a beginner,*

- I want to compile my source code.

- I want to navigate to my source code.

- I want to be able to edit my source.

- I want easy access to information about what I am doing.

- I want to install the tools I need to make code.

- I want to know what I can do.

- I want to be able to open everything like I was able in windows.

- I want to know what my command is going to do.

- I want the same visual feedback a file browser gives me.

- I want to know how to utilize my shells history.

- I want to be able to easily and obviously perform common tasks.

- when I type a filename, I want the most common task associate with that file to be suggested.

- when I drag a file into the command line, I want that file name to be typed.

*As a moderate user,*

- I want to automate my workflow.
- I want the system to alert me when automation is an option.

*As an expert user,*

- I want tab-completion that is file-type aware
- I want heads up information about my system as I use it

*As any user,*

- I want to be able to copy and paste.
- I want to be able to drag a link into my shell and have the file be added to the current directory.
- I want the system to learn about my most common tasks.
- I want to know what options are available to me for a given command.
- I want contextual information displayed to me when composing commands.
- I want to view files in the current directory as I move through the system.
- I want to be able to undo what I've done.

## 3.2   Hierarchical Analysis of Selected Tasks

1. As any user, I want to be able to undo what I've done.

   (a) Run a command

       i. Know what command to run
       ii. Write the command

       i. Understand the results of the command
          A. View what files/directories/settings were changed
          B. View how they were changed

   (b) Undo a command

2. As a beginner, I want to know what I can do with a file.

   (a) Specify a file

4

    i. See what files are in my current working directory

    ii. Select one of those files

(b) View the available commands

    i. View a list of what commands would be useful for that file

    ii. View what each command does

(c) Run a command on the file

    i. Select a command

3. As a casual user, I want to automate my workflow.

  (a) Specify a task which needs to be run many times

  (b) View suggestions on how to run that task

  (c) Select an automation technique

4. As a beginner, I want to know what the command is going to do.

  (a) Specify a command

  (b) View that commands effects

    i. View which files are going to be affected

    ii. View how those files are going to be affected

  (c) Chose whether to run the command or not

5. As an expert user, I want heads-up information about my system as I use it.

  (a) See what the state of my session is

    i. View background jobs

    ii. View possible commands based on the current context

      A. Take local files into account

    iii. View the ways the current session has been altered - non-normal environment variables, virtual environments, et cetera.

# 4 Task Navigation (Design Alternatives)

For each of the main tasks provided in Section **??**, we analyze the ways in which these tasks could be actualized. Each actualization is represented with three separate diagrams showing the flow of actions.

## 4.1 Undoing Actions

In this task, the user needs to be able to be able to revert an action they have just performed (something that is particularly useful for beginners to the command line environment).
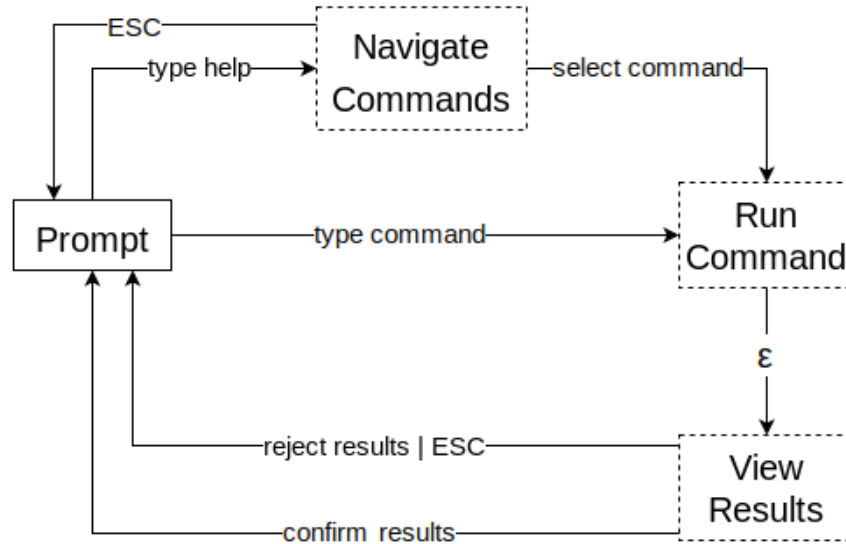
Figure 1: A breakdown of the procedure for undoing actions.

Figure 1 demonstrates a method which allows the user to see the results of their actions, and then chose whether or not to carry it out.
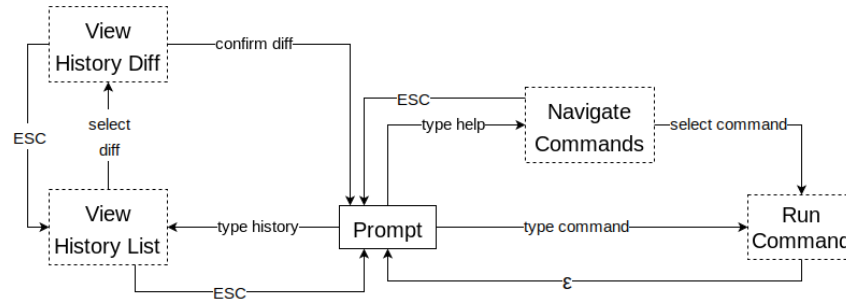


Figure 2: A breakdown of the procedure for undoing actions.

A scheme that gives perhaps more undo ability (yet less pedagogical value) is the ability to brows the history of your commands and the states they represented. Figure **??** shows such a scheme.
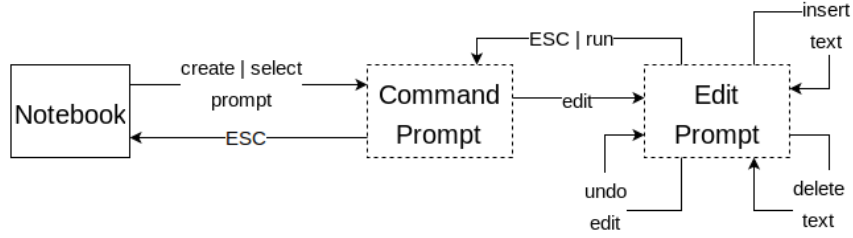
Figure 3: A breakdown of the procedure for undoing actions.

Another common way to deal with this problem is the style that is used by the *iPython notebook* — displaying the whole command history, and allowing the user to change the text of the commands that were previously entered. Figure 3 shows such a scheme.

## 4.2 Discovery

Users commonly want to know what they can do with given file types. We want to provide the user with the knowledge of how to continue with their files.
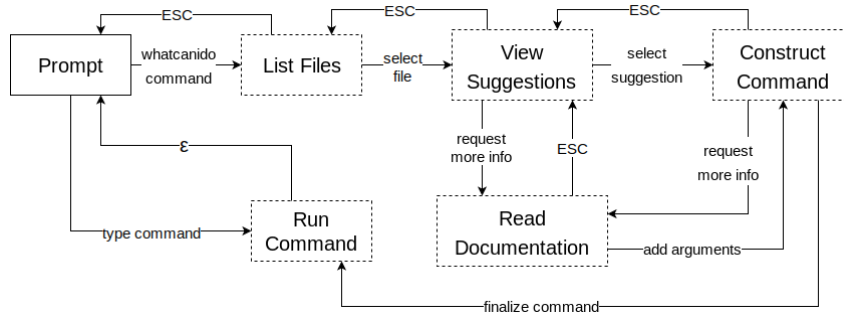


Figure 4: A breakdown of the procedure for undoing actions.

In Figure 4 we present a model where first a file is selected. Once the selection has been made, the shell can provide a list of "what to do next" options. This represents a more reverse polish style of shell navigation.

ESC | close pane

ESC | close pane

Notebook

select | create
a prompt

ESC

Command
Prompt

click a file

run | edit
prompt

View
File-Type
Synopsis

click
commands
link

View Possible
Commands

select a
command

request
more
info

ESC

finalize command

Construct
Command

request
more info

ESC

Read
Documentation

Read
Documentation
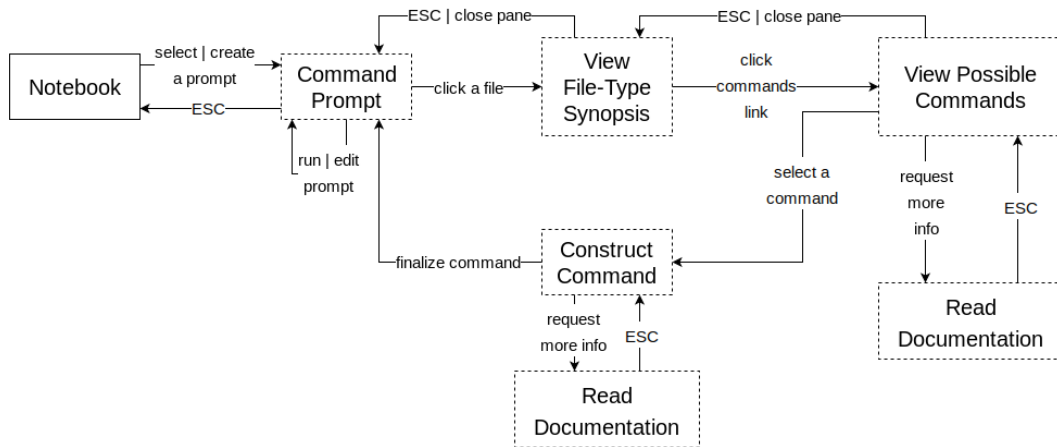
Figure 5: A breakdown of the procedure for undoing actions.

An alternative to this, again, would be a notebook style interface, with a mouse oriented approach. In this, we can select files, and can optionally be presented with a list of actions.

ESC

Prompt

whatcanido
command

View
Most-Frequent
Commands for
File Type

type a
command

ε

select a
command

Run
Command
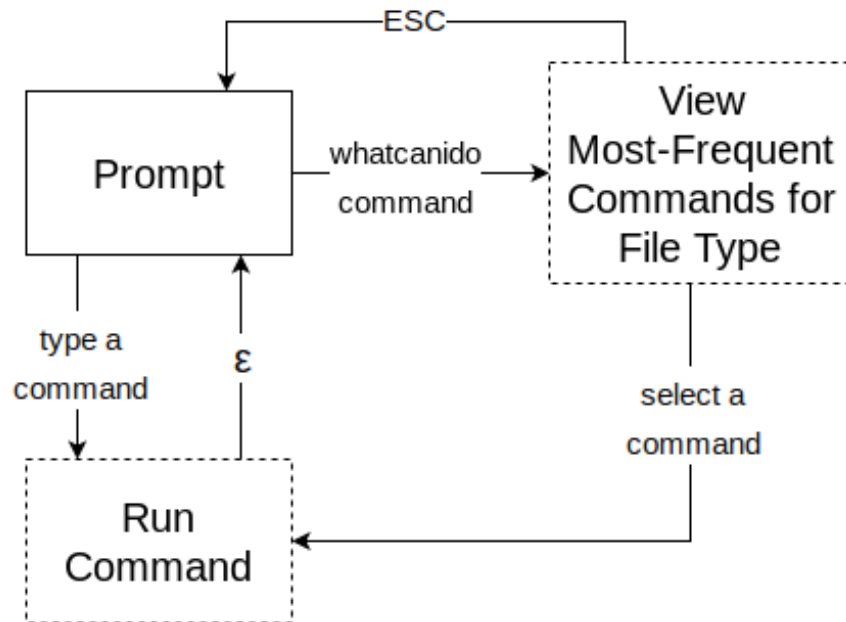
Figure 6: A breakdown of the procedure for undoing actions.

Figure 6 shows the simplest way to implement this feature — a special command to ask "what is the most common task to perform on this file type."

## 4.3 Workflow Automation

Commonly users are performing repetitive tasks on a related set of files. They may not understand that their workflow could be automated, or how to automate things. The onus would then be on our software to detect common situations as well as repetitive situations, and alert and assist the user in streamlining their work.
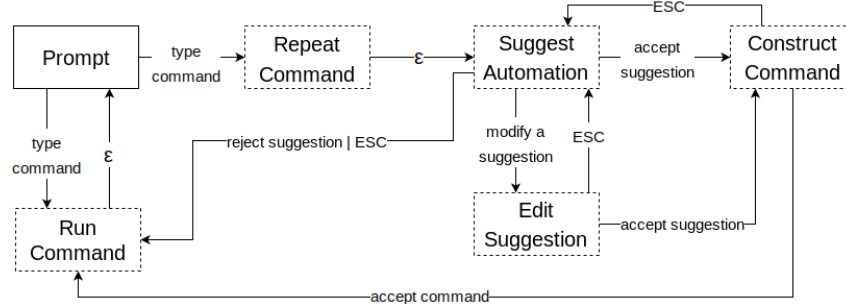


Figure 7: A breakdown of the procedure for undoing actions.

In Figure 7 we show perhaps the most obvious way to perform this sort of automation — detect repeated commands, and suggest an automation path (such as building a for loop). This will serve as a good, non-invasive method, that will alert the user of features of the shell.
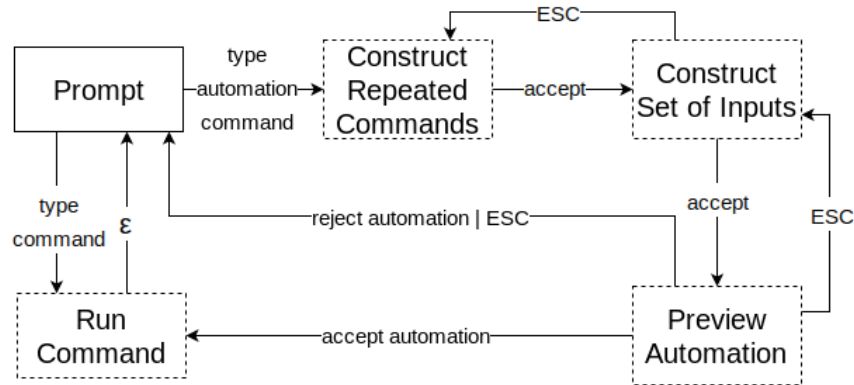


Figure 8: A breakdown of the procedure for undoing actions.

The user, however, could be provided with an "automate" command, prompting this system to provide automation suggestions. This would be an easier path to pursue, but leave more responsibility for the user. Figure **??** shows such a scheme.
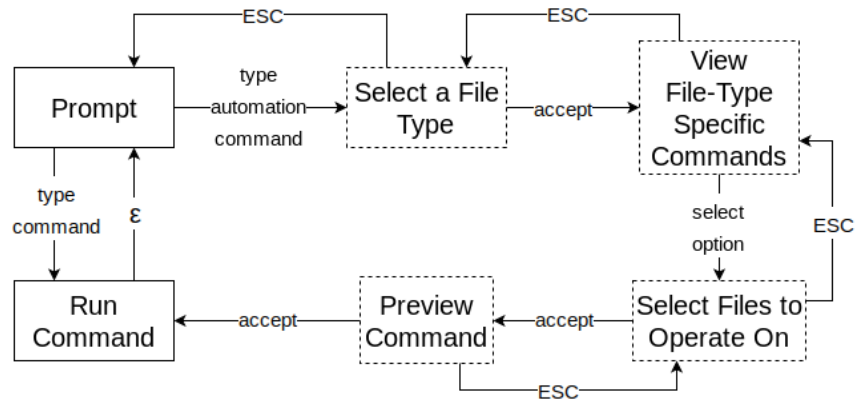
Figure 9: A breakdown of the procedure for undoing actions.

Finally, we could boil all automation commands down to batch processes on related files. Figure **??** shows how the user may select related files, and then build up a command to operate on all files.

## 4.4 Preview Command

Commonly users are not sure how what the results of their commands will be. We feel being provided with a "preview" of commands may help build confidence in the shell.
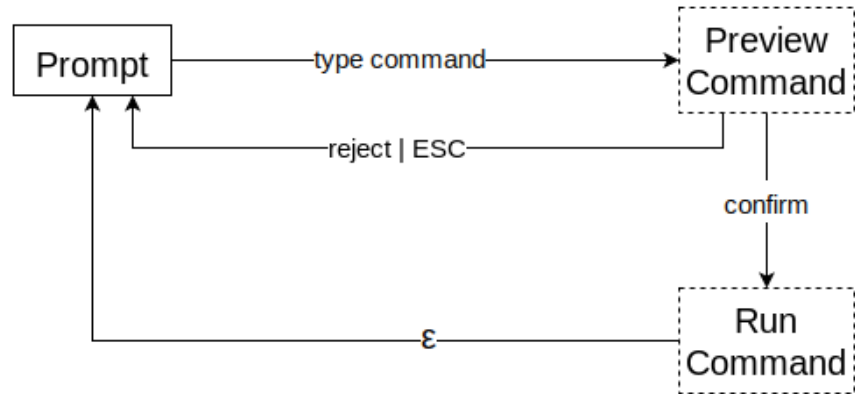


Figure 10: A breakdown of the procedure for undoing actions.

An easy way to achieve this would be to preview the command while it is being constructed, perhaps in the context window we have described. The user can then fluidly work in the shell, seeing results before they occur.
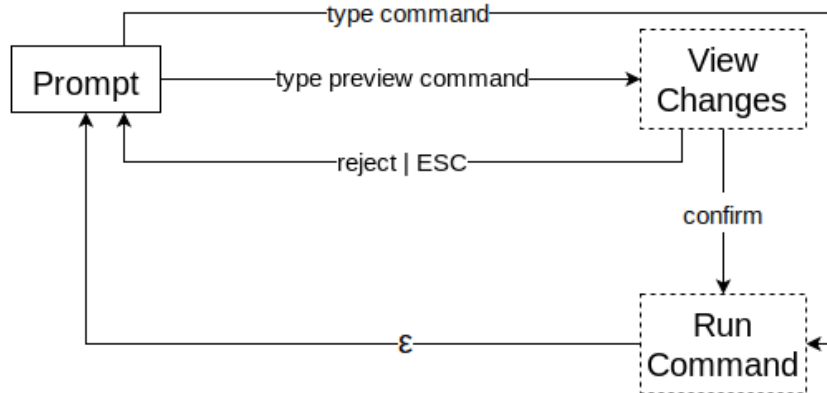
Figure 11: A breakdown of the procedure for undoing actions.

Figure ?? however would remove this as being the default behavior, and allow the users to request a preview. The rest of the workflow would remain unchanged.
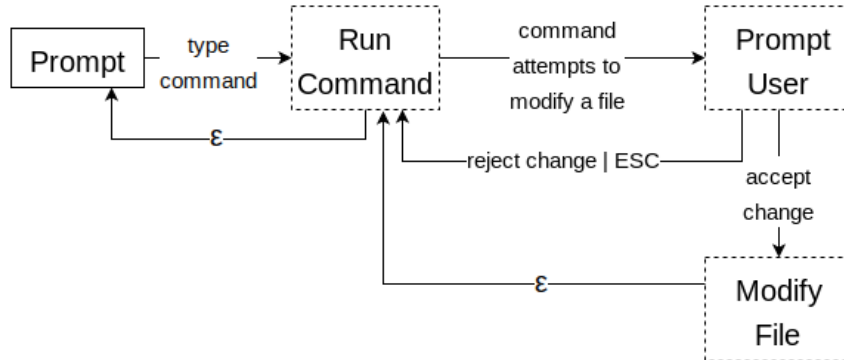


Figure 12: A breakdown of the procedure for undoing actions.

Finally, Figure ?? shows a case when previews are only used when the shell detects a chance will be made to any files (probably the time when users are most uncomfortable). This way the users are not constantly burdened with previews, and only shown then when they really matter.

# 5  Possible Interaction Techniques

As this is an addition to the CLI, and not a departure from it, we intend to keep the keyboard as the main focus. All new interfaces should be navigable by keyboard, if not least primarily so. For users new to a CLI, however, they can be assumed to be most comfortable with

interacting via a mouse. We imagine that for the novice and casual user alike, a hybrid interface could be highly useful.

### The Gestalt Principles

Proximity – In the shell there is a fixed amount of space between lines and individual characters is fixed. However, textual output can still take advantage of the principle of proximity by using whitespace to separate items that should not be grouped together. This achieves the goal of grouping related objects in a text-based interface.

Similarity – Text mainly conveys semantic information, and we can use a few techniques to visually tag that information so that the user can easily determine the type of the text theyre reading and writing. Specifically, automatically making use of output coloring and syntax highlighting can help CLI a user pick apart information and group similar words within the buffer.

Past Experience – Most importantly, we need to appeal to the experiences of both novice and advanced users. Those who have never used the shell will expect their filesystem to be presented to them in a way that reminds them of a file browser - which is why we want to include a graphical pane which can be used to fill in commands equivalent to point and click actions. More advanced users will be more familiar with the basics of common commands, and can benefit from command completion that will make it easier to type the things that they type most often.

Color and light patterns could be used to convey information about the current session. In class the structure of eyes was discussed, with a focus on what sort of light is most easily noticed by different sections of they eye. In a text-based interface this theory could be used to craft notifications that catch the users attention based on the distance of the notification from the users likely focus on the screen.

## 6  Existing Systems

While there exists no application that cover all of the contextual and adaptive features discussed in this paper, the following existing systems tackle many of our ideas for a richer shell.
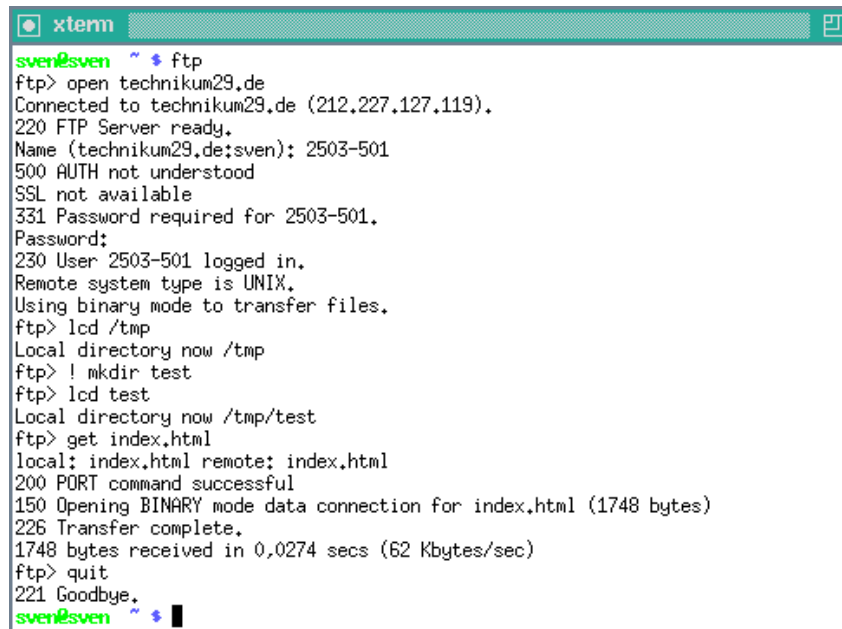
### 6.1  xterm

While there exists no application that covers all of the contextual and adaptive features discussed in this paper, the following existing systems tackle many of our ideas for a richer shell.

Figure 13: An xterm window.

## 6.2   iterm2

iterm2, like xterm, essentially just provides a terminal emulator, which in turn provides a shell. However, it is able to provide far more contextual information than that of xterm. The shell can alert the terminal emulator itself about the state of the system (such as current git status), as well as embed images into the text result.

## 6.3   zsh / Fish

Both zsh and fish are shell programs designed to be more usable alternatives to bash. A large part of this usability revolves around autocompletion and suggestion. Fish will suggest command as you type (though it is not bash-compatible). Zsh allows arbitrary autocompletion to be added to the system.

## 6.4   Explain Shell

Explain shell is an educational tool which will break up a command, and explain how it worked and what it means. This tool does not itself provide a shell, but it does provide a tool which can be used to learn and understand bash and related programs.
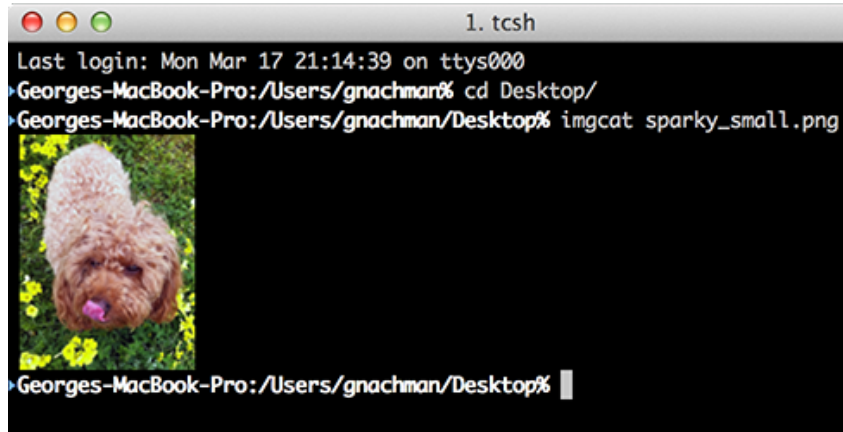
Figure 14: An iTerm 2 window, with inline images.

## 6.5 Jupyter

Project jupyter is a modular, web based shell, designed for use with data analytic languages (such as Julia, Python, and R). It allows graphical response to commands, and lets users distribute entire sessions of use. An important feature is once a command is run, the user can edit the command. This chance will then propagate through the history.

## 6.6 Comparison of Features

|  | This | xterm | iTerm 2 | zsh/fish | Explain Shell | Jupyter |
|---|---|---|---|---|---|---|
| Undo | V | X | X | X | X | V |
| File Actions | V | X | X | O | X | O |
| Automation Assistance | X | X | X | X | X | X |
| Command Results | V | X | X | X | V | O |
| Heads-Up Information | V | O | V | V | X | V |

Table 1: Comparision of features. V – able to perform the task, X – unable to perform the task, O – able to perform the task with poor interactive design.

Figure 15: A fish shell, showing a suggested input.
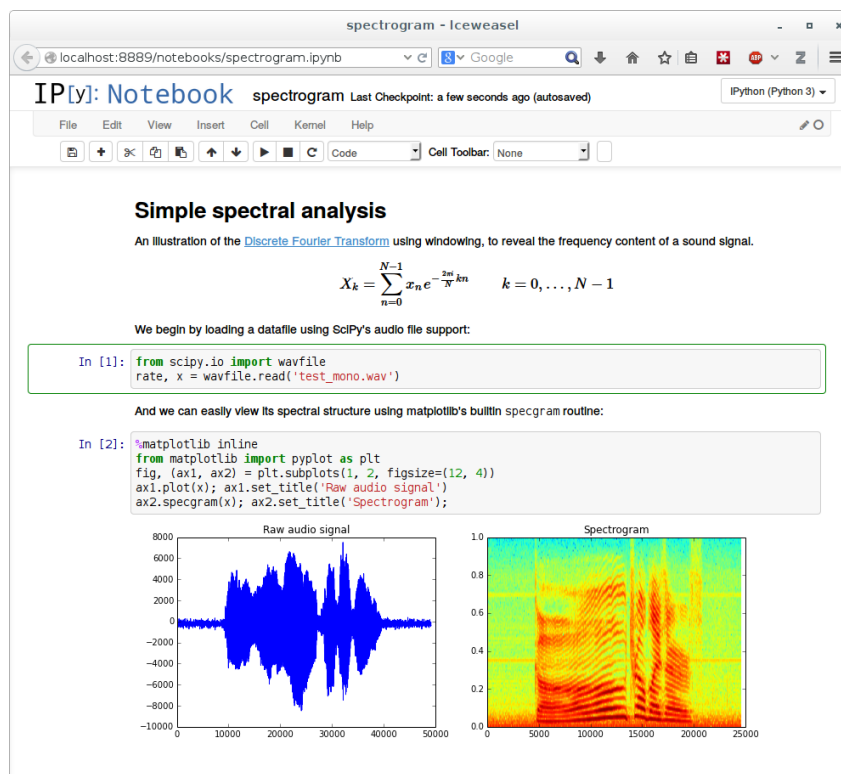


Figure 16: A screen shot of explain shell breaking down a command.

Figure 17: An interactive jupyter session.