Reimagining the Command Line in the Tablet Age

R. Saravanan
Texas A&M University

sarava@mindmeldr.com

TexasLinuxFest, San Antonio, August 4, 2012

A GUI gives you sentences you can say to express yourself.

A command line interface gives you words.

Anonymous

Limitations of the Command-Line Interface (CLI)
as compared to the Graphical User Interface (GUI)

- "Newbies" find a blank terminal screen confusing
  - Steep learning curve (need to learn basic commands first)
  - Abbreviated commands (efficient, but cryptic!)
  - Command have to be typed precisely (unlike natural language)
  - Poor feedback on the results of commands; *difficult to undo*
  - A GUI with good icons is more intuitive
    - Relies upon **recognition** rather than **recall**
- Looks dull and archaic (does not use rich monitor display)
  - Sometimes a picture is worth a thousand words
- Does not use the analog input capabilities of the mouse

Advantages of the CLI

- More powerful and flexible than the GUI
  - Not limited to actions that can be fitted on the screen
  - Pipe output of one command to another
    - i.e., form your own sentences using words!
  - Wild-carding, command completion, history recall

- Efficient use of screen real-estate
  - Easy to use on small screens and remotely

  (Mobile phones tend to use a menu-driven text UI, although not a CLI)

- Self-documenting
  - Useful for scripting and automation, especially on the server-side

XMLTerm (Mozilla; ca. 2000)

AjaxTerm (HTML+JS+CSS; ca. 2006)

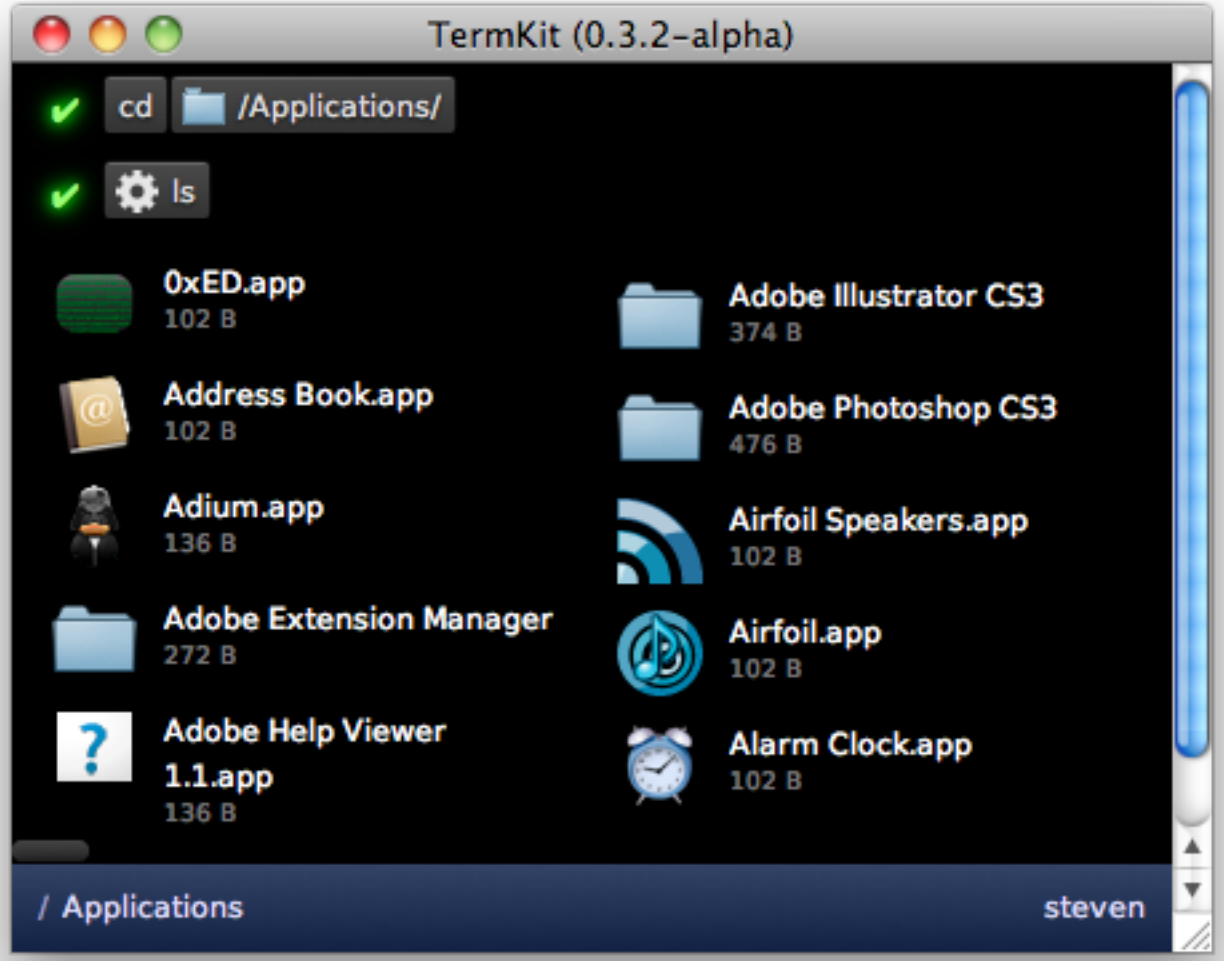
File Edit View Go Bookmarks Tools Help

http://localhost:8080/

Colors GET Paste

```
*help.txt*      For Vim version 6.4.  Last change: 2004 Aug 29

                        VIM - main help file
                                                                  k
      Move around:  Use the cursor keys, or "h" to go left,     h   l
                    "j" to go down, "k" to go up, "l" to go right.  j
Close this window:  Use ":q<Enter>".
   Get out of Vim:  Use ":qa!<Enter>" (careful, all changes are lost!).

Jump to a subject:  Position the cursor on a tag between |bars| and hit CTRL-].
   With the mouse:  ":set mouse=a" to enable the mouse (in xterm or GUI).
                    Double-click the left mouse button on a tag between |bars|.
        Jump back:  Type CTRL-T or CTRL-O (repeat to go further back).

Get specific help:  It is possible to go directly to whatever you want help
                    on, by giving an argument to the ":help" command |:help|.
                    It is possible to further specify the context:
                                                        *help-context*
                          WHAT                  PREPEND    EXAMPLE
                      Normal mode command      (nothing)   :help x
help.txt [help][RO]                                          1,1           Top

~
[No File]                                                    0,0-1         All
```

TermKit (Webkit; 2011)



GraphTerm: A Graphical Terminal Interface

- Aims to seamlessly blend the CLI and the GUI
- Fully backwards-compatible terminal emulator for *xterm*.
  - Use it just like a regular terminal interface, accessing additional graphical features only as needed
- Builds upon two earlier projects, XMLTerm and AjaxTerm
- Alpha quality (*dogfood status for the past 3 weeks!*)

- Project Page

  http://info.mindmeldr.com/code/graphterm

- Source code (BSD License)

  https://github.com/mitotic/graphterm

Installing and running GraphTerm

- Install
  - `sudo apt-get install –y python-setuptools`
  - `sudo easy_install graphterm`
  - `sudo gterm_setup`

- Start server
  - `gterm_server –auth_code=none`

- Open terminal in browser (to connect to localhost)
  - `http://localhost:8900`
  - Terminal sessions have URLs of the form:
    `http://localhost:8900/<hostname>/tty1`
  - First user to open the URL owns the terminal session, and others can use the same URL to watch it (or steal it)

- Connect additional hosts to server
  - `gterm_host <servraddr> <hostname>`

GraphTerm Features

- Backwards compatible with CLI, plus incremental feature set
- Clickable (hyperlinked) text for filenames and commands
  - *Adaptive* paste behavior (depending upon the current command line)
- Optional icons for file listings etc.
- Platform-independent client (HTML+Javascript)
  - Themable using CSS
- Pure python server (should work on any Unix-ish system)
- Touch-friendly
  - Translate clicks, taps, and drops into textual commands
- Cloud and collaboration-friendly
  - Single (tabbed) browser window to connect to multiple hosts
  - Drag and drop to copy files between different hosts
  - **Screen sharing**: A terminal session can be shared by multiple users

GraphTerm Implementation

- Server: ~4500 lines of python
  - Tornado webserver, code from AjaxTerm
  - Websocket (2-way HTTP) connections between browser and server
- Client: ~2500 lines of HTML+JS+CSS
  - jQuery, AJAX editor (ACE)
- Adds a new *xterm* “escape” sequence to switch to a **graphical screen mode** to display HTML fragments (“pagelets”)
  - GraphTerm-aware programs using this mode can be written in *any* language: **gls, gvi, …**
- Browser connects to GraphTerm server using websockets
  - Host computers where the terminal session runs also connect to the same server (on a different port)

ls vs. gls

local/tty2/

localhost:8900/local/tty2/

```
graphterm$ ls
__init__.py          gterm.py             lineterm.py          packetserver.py
__init__.pyc         gterm.py~            lineterm.pyc         packetserver.pyc
__init__.py~         gterm_setup.py       lineterm.py~         packetserver.py~
bin                  gterm_setup.py~      ordereddict.py       testsslclient.py
certs                gtermserver.py       ordereddict.pyc      testsslclient.py~
docs                 gtermserver.pyc      otrace.py            tornado
episode4.txt         gtermserver.py~      otrace.pyc           www
graphterm$ gls
```

..	.	~	
__init__.py	__init__.pyc	__init__.py~	bin
certs	docs	episode4.txt	gterm.py
gterm.py~	gterm_setup.py	gterm_setup.py~	gtermserver.py
gtermserver.pyc	gtermserver.py~	lineterm.py	lineterm.pyc
lineterm.py~	ordereddict.py	ordereddict.pyc	otrace.py
otrace.pyc	packetserver.py	packetserver.pyc	packetserver.py~
testsslclient.py	testsslclient.py~	tornado	www

```
graphterm$
```

Checking the weather using Google Weather API

local/tty2/

localhost:8900/local/tty2/

```
graphterm$ gweather College Station
```

Current weather in College Station, TX


88 °F, Clear

Forecast


Tue: Chance of Storm, Low 75 °F, High 97 °F


Wed: Mostly Sunny, Low 77 °F, High 97 °F


Thu: Mostly Sunny, Low 75 °F, High 97 °F


Fri: Mostly Sunny, Low 77 °F, High 100 °F

```
graphterm$
```

Collapse command output (stdout)

local/tty2/

localhost:8900/local/tty2/

Session: local/tty2/control Actions Icons: on Webcast: off Theme: default

Home Bottom Collapse Expand Detach New Help About

```
www$ cd ..
graphterm$ cd www; gls -f
www$
www$ cd ..
graphterm$ cd www; gls -f
www$ ls -l images
www$ gls
www$ ls images
www$ gls
www$ ls images
www$
www$ ls
www$
```

Control Top ▲ ▼ ⌫ < > Clear Cmd Opt File Enter

CSS-themable (3D perspective)

Graphical editing in the “cloud” using *gvi*

localhost:8900/local/tty2/

Save Discard

```
187
188     def remote_terminal_update(self, term_name=None, add_flag=True):
189         """If term_name is None, generate new terminal name and return it"""
190         if not term_name:
191             while True:
192                 self.term_count += 1
193                 term_name = "tty"+str(self.term_count)
194                 if term_name not in self.term_set:
195                     break
196
197         if add_flag:
198             self.term_set.add(term_name)
199         else:
200             self.term_set.discard(term_name)
201         return term_name
202
203     def remote_response(self, term_name, msg_list):
204         ws_list = GTSocket._watch_set.get(self.connection_id+"/"+term_name)
205         if not ws_list:
206             return
207         for ws_id in ws_list:
208             ws = GTSocket._all_websockets.get(ws_id)
209             if ws:
210                 try:
211                     ws.write_message(json.dumps(msg_list))
212                 except Exception, excp:
213                     logging.error("remote_response: ERROR %s", excp)
214                     try:
215                         # Close websocket on write error
216                         ws.close()
217                     except Exception:
218                         pass
```

Split-screen scrolling

local/tty2/

localhost:8900/local/tty2/

```
tango-audio-x-generic.png                tango-text-x-script.png
tango-folder.png                         tango-video-x-generic.png
www$ gls

..                .                      ~
ace.js            graphterm.css          graphterm.js           images
index.html        index.html~            jquery                 mode-css.js
mode-css.js~      mode-html.js           mode-html.js~          mode-javascript.js
mode-python.js    mode-xml.js            mode-xml.js~           perspective.css
perspective.css~  test1.html             test1.html~            theme-twilight.js
theme-vibrant_ink.js  themes
www$ ls images
```

Control Top ▲ ▼ ⌫ < > Clear Cmd Opt File Enter

Emacs in the browser

local/tty2/

localhost:8900/local/tty2/

```
ef remote_terminal_update(self, term_name=None, add_flag=True):
    """If term_name is None, generate new terminal name and return it"""
    if not term_name:
        while True:
            self.term_count += 1
            term_name = "tty"+str(self.term_count)
            if term_name not in self.term_set:
                break

    if add_flag:
        self.term_set.add(term_name)
    else:
        self.term_set.discard(term_name)
    return term_name

ef remote_response(self, term_name, msg_list):
    ws_list = GTSocket._watch_set.get(self.connection_id+"/"+term_name)
    if not ws_list:
        return
    for ws_id in ws_list:
--F1  gtermserver.py    16% L198    (Python)----------------------------------------
```

Potential Applications for GraphTerm

- A more fun (and powerful) replacement for the *terminal*
- A replacement for *GNU screen* (portable, detachable terminal sessions)
- Manage a collection of computers using the browser
  - Debug Python programs using ***otrace***
  - **Audit history** of all commands
- For collaboration with other developers
  - Screen sharing and screen stealing
- For demonstrating or teaching command-line based software
  - Create a **virtual computer lab** using the cloud
  - Interact using screen sharing and twitter

The End