wSSH

Final Presentation

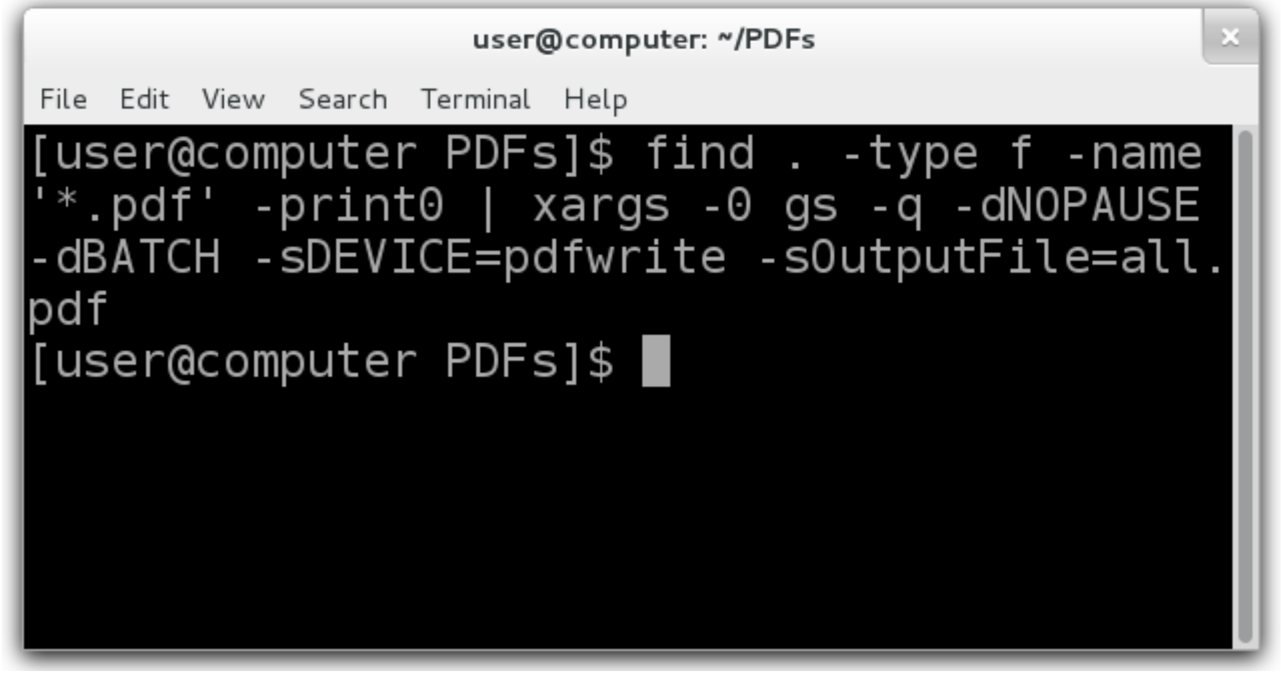
James McGinnis, David Johnston*, Kerrick Staley*, Jonathon Saunders, Cyle Dawson

*delivered Jumpstart Presentation

Problem Statement

CLIs are powerful

Command line interfaces allow concise expression of complex operations



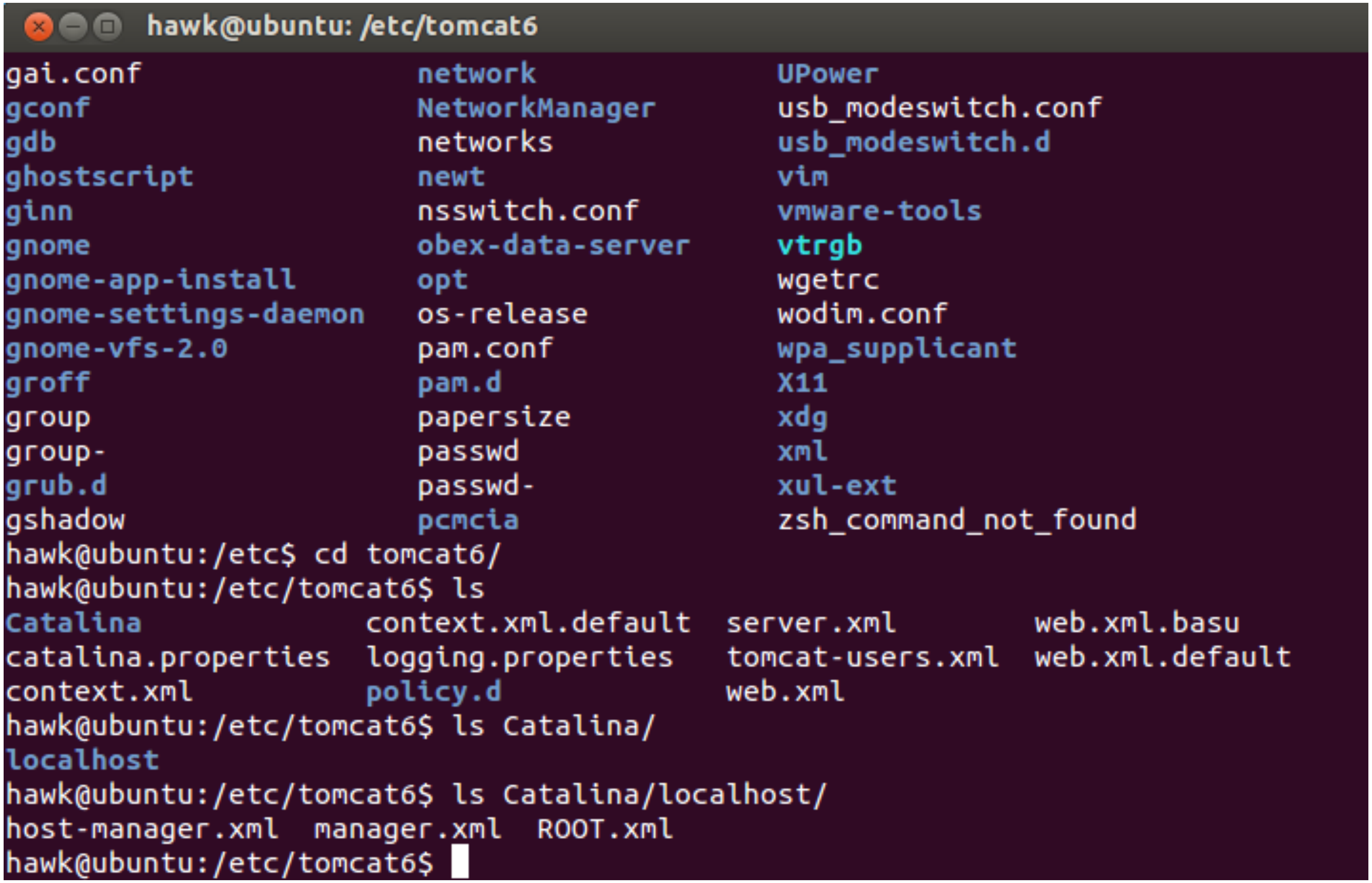
```
[user@computer PDFs]$ find . -type f -name
'*.pdf' -print0 | xargs -0 gs -q -dNOPAUSE
-dBATCH -sDEVICE=pdfwrite -sOutputFile=all.
pdf
[user@computer PDFs]$
```

But basic operations are cumbersome

Consider:

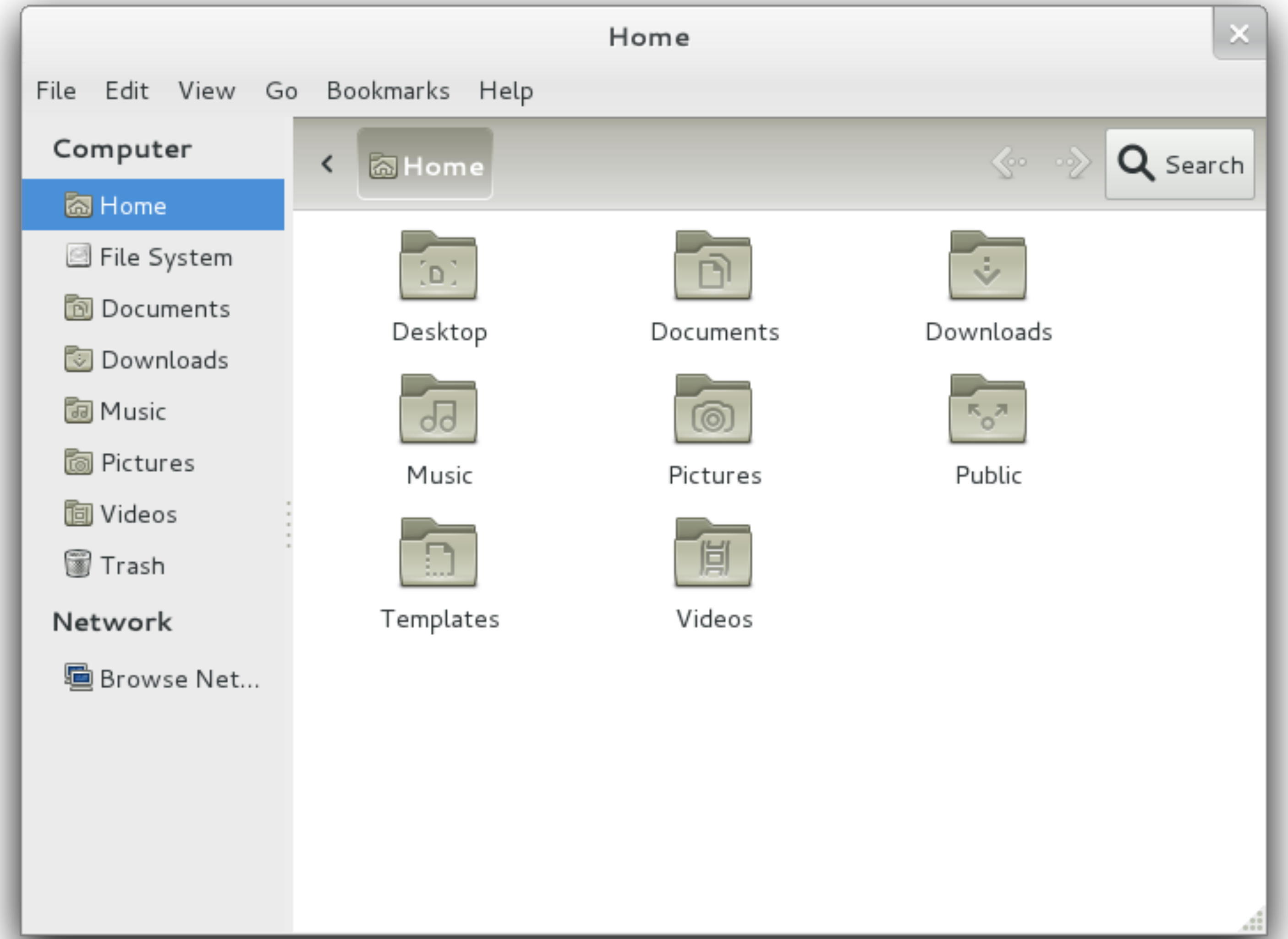
- **`rm`**
- **`mv`**
- **`cp`**
- **`ls`**

It can be hard to navigate complex file-system via CLI.



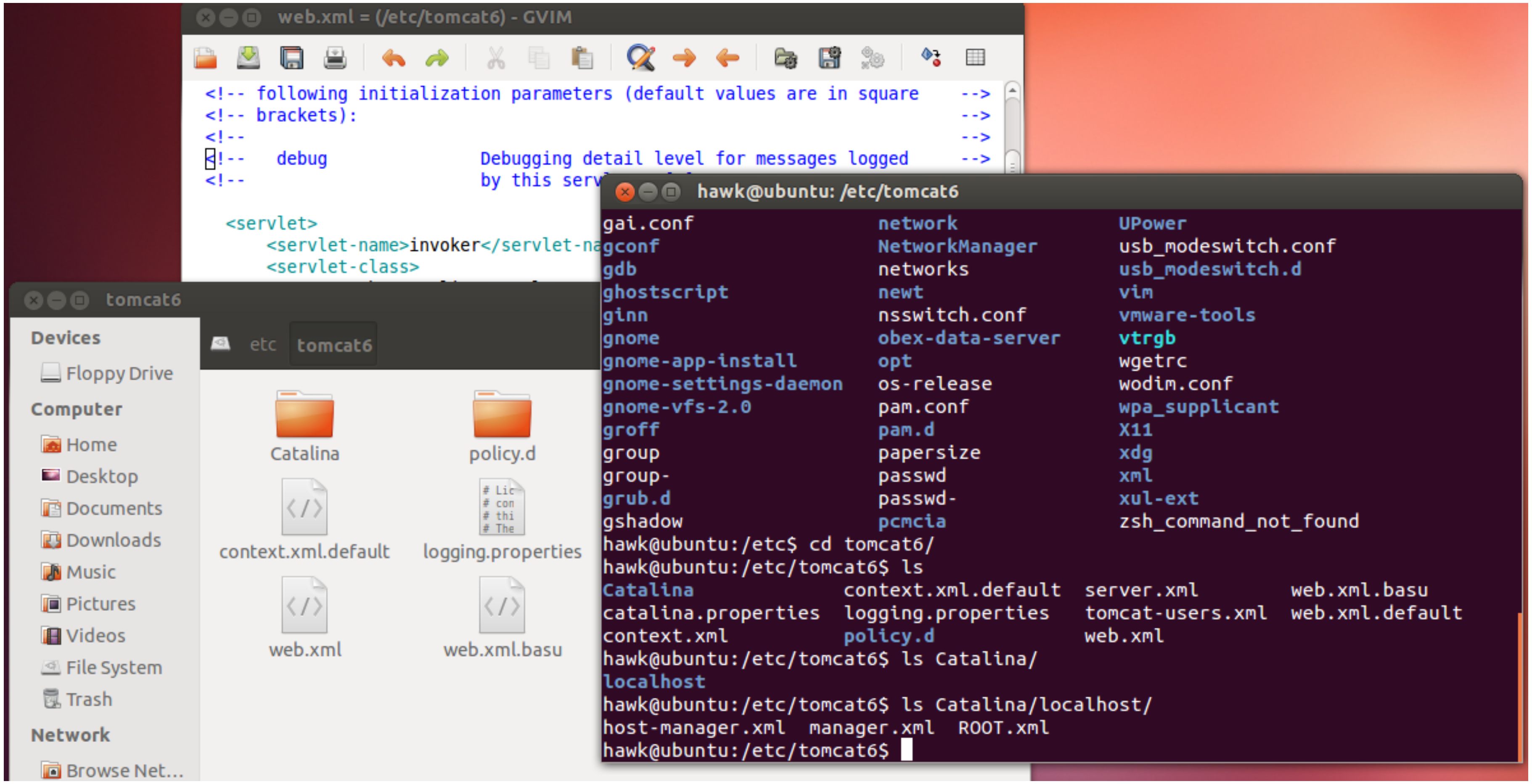
```
gai.conf                    network               UPower
gconf                       NetworkManager        usb_modeswitch.conf
gdb                         networks              usb_modeswitch.d
ghostscript                 newt                  vim
ginn                        nsswitch.conf         vmware-tools
gnome                       obex-data-server      vtrgb
gnome-app-install           opt                   wgetrc
gnome-settings-daemon       os-release            wodim.conf
gnome-vfs-2.0               pam.conf              wpa_supplicant
groff                       pam.d                 X11
group                       papersize             xdg
group-                      passwd                xml
grub.d                      passwd-               xul-ext
gshadow                     pcmcia                zsh_command_not_found
hawk@ubuntu:/etc$ cd tomcat6/
hawk@ubuntu:/etc/tomcat6$ ls
Catalina               context.xml.default  server.xml         web.xml.basu
catalina.properties    logging.properties   tomcat-users.xml   web.xml.default
context.xml            policy.d             web.xml
hawk@ubuntu:/etc/tomcat6$ ls Catalina/
localhost
hawk@ubuntu:/etc/tomcat6$ ls Catalina/localhost/
host-manager.xml  manager.xml  ROOT.xml
hawk@ubuntu:/etc/tomcat6$ 
```

Sometimes the simplicity of a GUI file-browser is just more convenient.



Target Users

Target User: Anyone who has needed to use a CLI and GUI together



```
<!-- following initialization parameters (default values are in square -->
<!-- brackets):                                                          -->
<!--                                                                     -->
<!--   debug               Debugging detail level for messages logged    -->
<!--                       by this serv
  <servlet>
      <servlet-name>invoker</servlet-na
      <servlet-class>
```

```
gai.conf                  network                 UPower
gconf                     NetworkManager          usb_modeswitch.conf
gdb                       networks                usb_modeswitch.d
ghostscript               newt                    vim
ginn                      nsswitch.conf           vmware-tools
gnome                     obex-data-server        vtrgb
gnome-app-install         opt                     wgetrc
gnome-settings-daemon     os-release              wodim.conf
gnome-vfs-2.0             pam.conf                wpa_supplicant
groff                     pam.d                   X11
group                     papersize               xdg
group-                    passwd                  xml
grub.d                    passwd-                 xul-ext
gshadow                   pcmcia                  zsh_command_not_found
hawk@ubuntu:/etc$ cd tomcat6/
hawk@ubuntu:/etc/tomcat6$ ls
Catalina              context.xml.default  server.xml         web.xml.basu
catalina.properties   logging.properties   tomcat-users.xml   web.xml.default
context.xml           policy.d             web.xml
hawk@ubuntu:/etc/tomcat6$ ls Catalina/
localhost
hawk@ubuntu:/etc/tomcat6$ ls Catalina/localhost/
host-manager.xml  manager.xml  ROOT.xml
hawk@ubuntu:/etc/tomcat6$
```

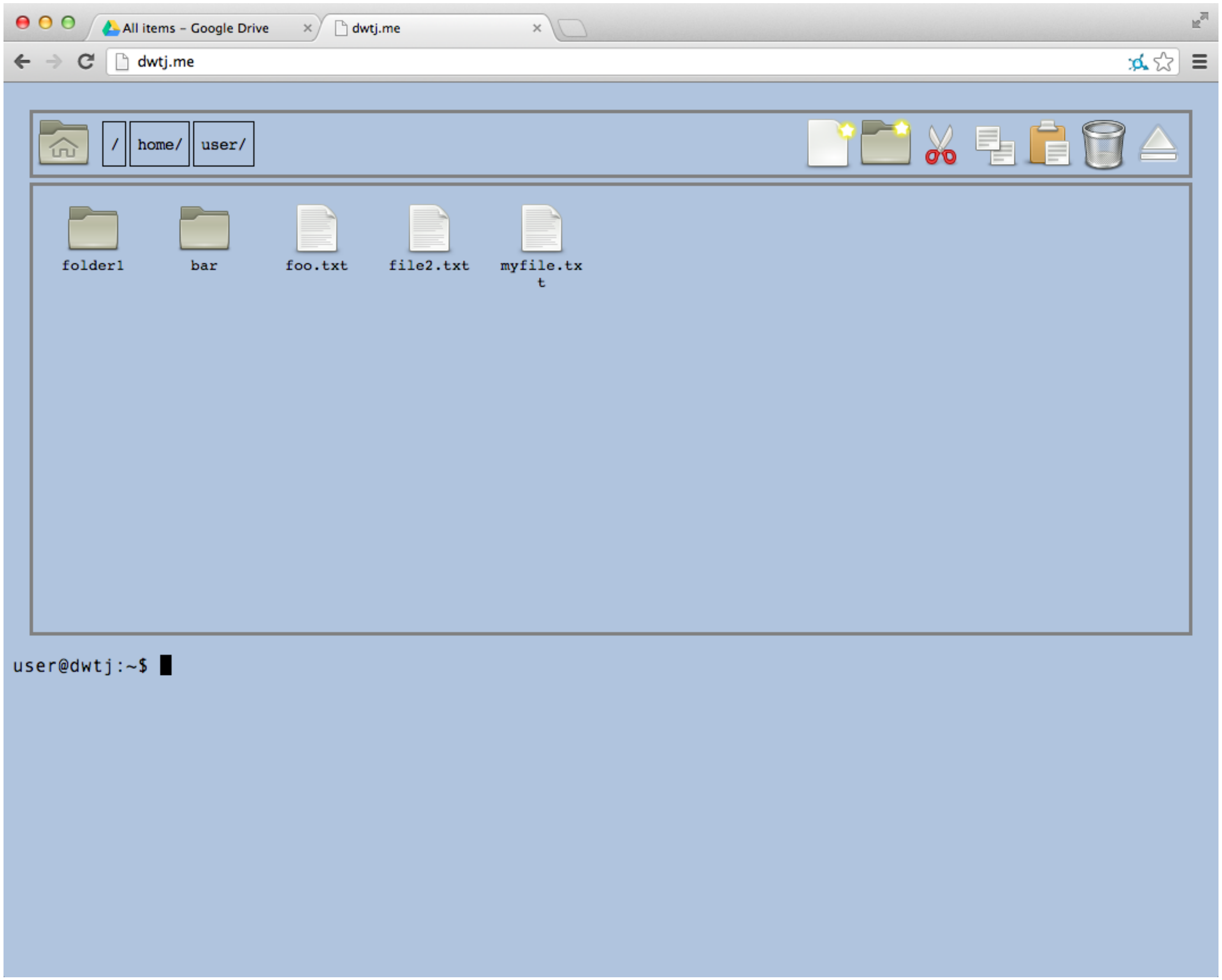
Our Solution

Our Solution

To make a better SSH client:

- Combine a GUI file browser with a remote terminal connection.
- Make them work together.

wSSH

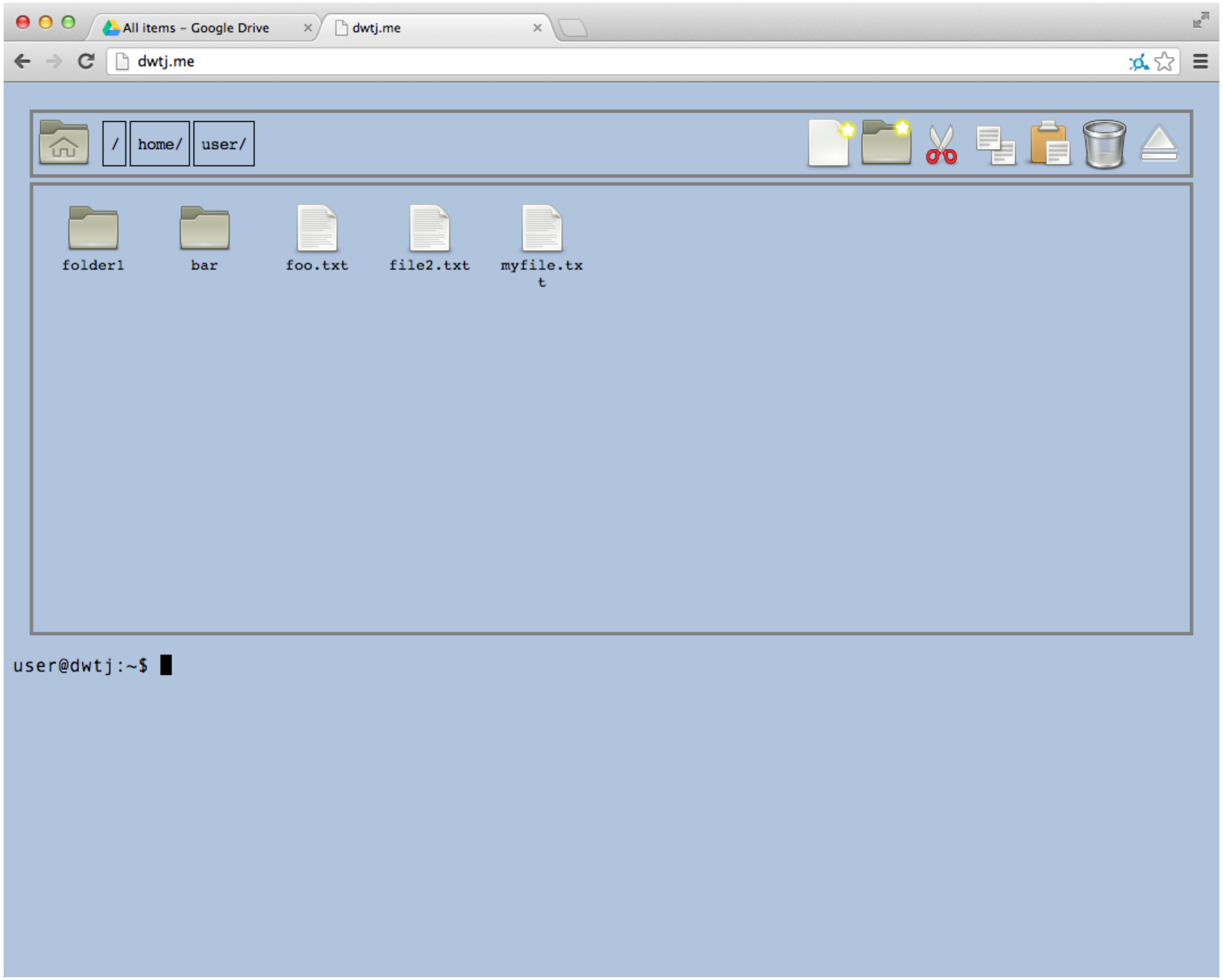


```
user@dwtj:~$ █
```

Our Solution

- Let the SSH user view and graphically manipulate files in their working directory using a simple file browser.
- Run in a browser, allowing access from anywhere.

Key Feature: Keep CLI and GUI in Sync



```
user@dwtj:~$ █
```

Key feature: Keep CLI and GUI in-sync

- Let the user decide which tasks should be done with each UI paradigm.
- Seamlessly transition from one paradigm

Use Cases and Usability Features

Use Case: Navigate in GUI, Work in CLI

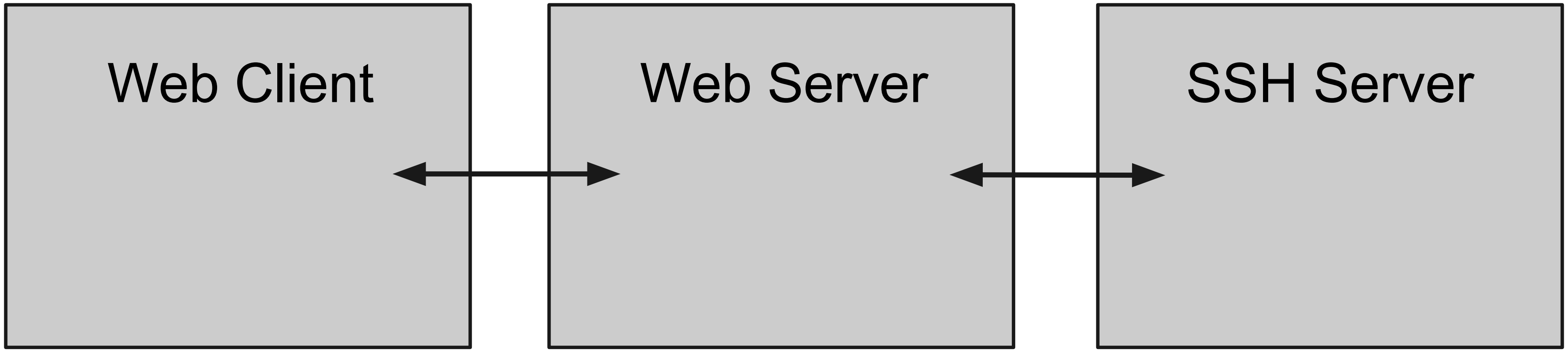
- User locates some directory via GUI
- Working directory in CLI mirrors with `cd`.
- User performs operations on files in that directory using CLI

Use Case:
Work in CLI, Occasionally Use GUI

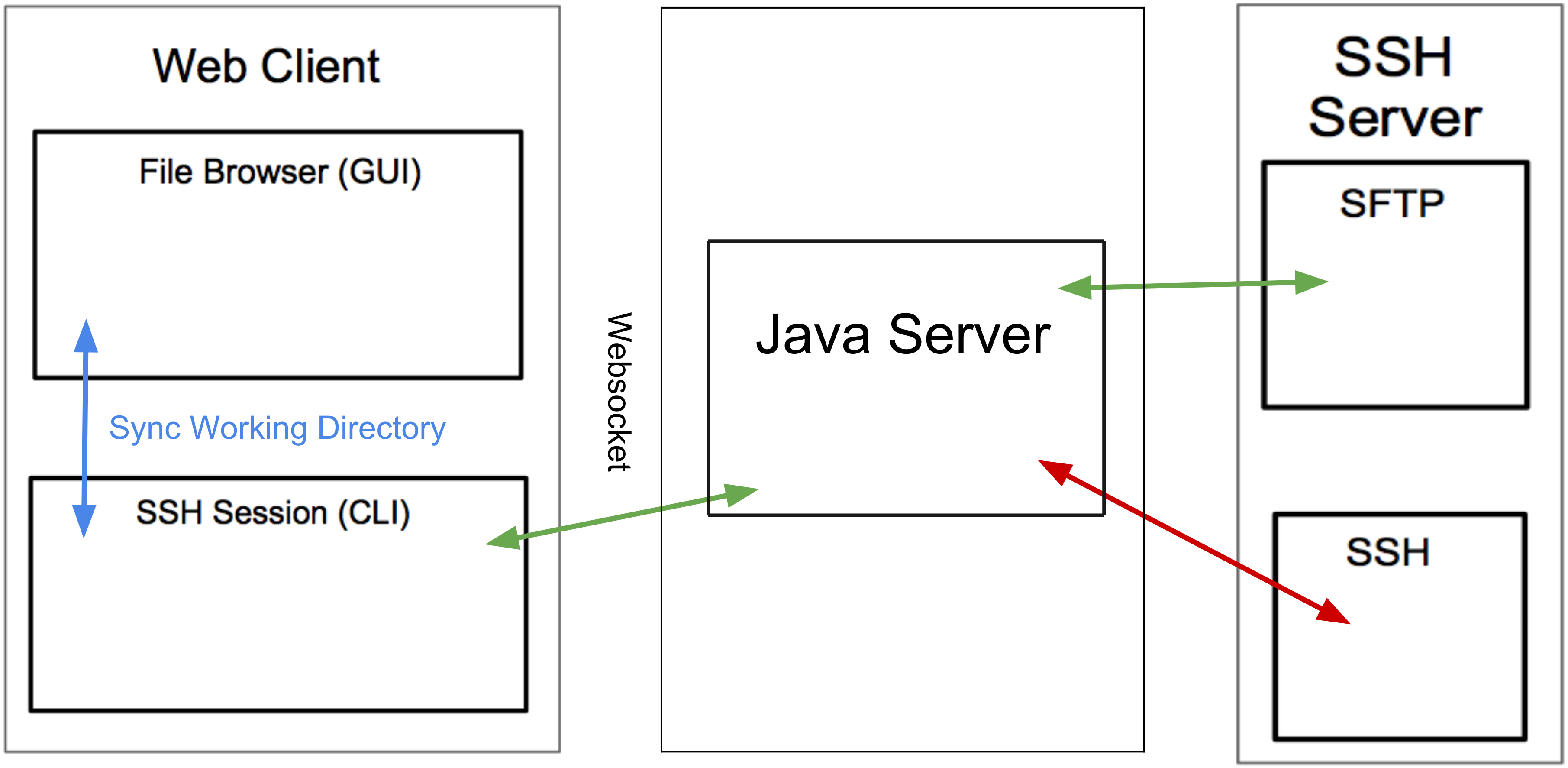
While performing various operations in CLI, the user selects and deletes a miscellaneous set of files via the GUI, because it seemed easier than using **`rm`**.

Implementation

Architectural View: 3 Hosts



Architectural View: Modules



Web Client Languages and Frameworks

Languages and Frameworks: Client GUI

- JavaScript
  - JQuery
  - JQuery UI Library
  - JSON
- Shell in a Box
  - Web-based terminal emulator
  - Sends JSON objects

Why JQuery?

Easily create a dynamic gui:

- Dynamically generate UI elements.
- Dynamically set and modify event handlers.

jQuery
write less, do more.

Why JQuery UI?

Easily create a highly interactive GUI.

Provides APIs to easily implement common GUI conventions.

- Selectable
- Draggable
- Droppable

JQuery UI: Event Abstraction

Some low-level javascript events:

- **`onclick()`**
- **`onmousedown()`**
- **`onmouseup()`**
- **`onmouseover()`**
- **`onmouseout()`**

From these, JQuery UI provides useful event abstractions such as

- **`selected()`**
- **`unselected()`**
- **`drag()`**
- **`drop()`**

Web CLI: Shell in a Box

- Knows how to emulate a terminal.
- Open source library of which we used part, **`vt100.js`**
- It was neatly abstracted and modularized

Web Server Languages and Frameworks

SSH

Authenticated, encrypted remote terminal client and server.

Java

We used Java on the webserver to make connections to SSH and SFTP and relay requests from and replies to the client.

Did someone mention PHP?

Last time we told you that we were going to use PHP (phpseclib) to handle the SSH connection.

What went wrong?

- For each user we wanted a server that would
  - Have a thread running continuously for each user.
  - Keep a persistent connection to the SSH server.
  - Wait *indefinitely* for user events.
- This wasn't really possible with PHP, since
  - PHP scripts aren't meant to block
  - By design, PHP will eventually timeout.

Websockets

- Provide a persistent connection between web clients and a "ws://" server.
- Different from ajax because bidirectional communication (Both client and server can push.)

Fortunately, SSH is a big deal.

SSH clients have been implemented in many other languages:

- C/C++
- Perl
- Python
- Java
- ...

Java? But why?

Our familiarity and its Libraries

- JSch
  - SSH/SFTP library.
  - can connect to an SSH/SFTP server from Java!
- java-websockets
  - An open connection to the web client.
  - Can extend server functionality without having to worry about the details of websockets.
  - Can use event-driven paradigm.

Communication Protocol

Responsibilities

Web Client

- File-browser (GUI): David and Jon
- Terminal (CLI): Kerrick

Server

- Client Interface: Cyle
- SSH/SFTP Interface: James

First Timeline

Sept. 24 - Oct. 8:

Create use-case diagrams and detailed UI mockups; prepare jumpstart presentation

Oct. 8 - Nov. 5:

Prototype individual modules - GUI, CLI, backend

Nov. 5 - Nov. 19:

Integrate frontend/backend

Test everything

Improve efficiency/scalability

Nov. 19 - Dec. 7

Polish and prepare final presentation

Actual Timeline

Sept. 24 - Oct. 8:

Create use-case diagrams and detailed UI mockups; prepare jumpstart presentation

Oct. 8 - Nov. 5:

Began building individual modules and finalizing design.

Realize that our architecture is flawed.

Nov. 5 - Nov. 19:

Finish new architecture design with Java instead of PHP

Begin Programming

Nov. 19 - Dec. 7

Finish Programming and put the pieces together.

Future Work

- Security
  - SSL/TLS of the websocket connection.
  - More security considerations.
- UI Features
  - Let user collapse either GUI or CLI
  - File uploads and downloads
  - Keyboard Shortcuts
- Performance/Architecture
  - Intelligently buffer communications.
  - Push changes to the current working directory rather than polling for them.

Demo

Questions?

Client Event Generation

JSON-encoded requests/responses are sent to and from the client and server over the websocket.

```
{"keys": "e"}

{"mv": [ [ "/home/user/file1.txt",
           "/home/user/file2.txt" ],
         "/home/user/directory/" ]
}
```

CGI vs Sockets

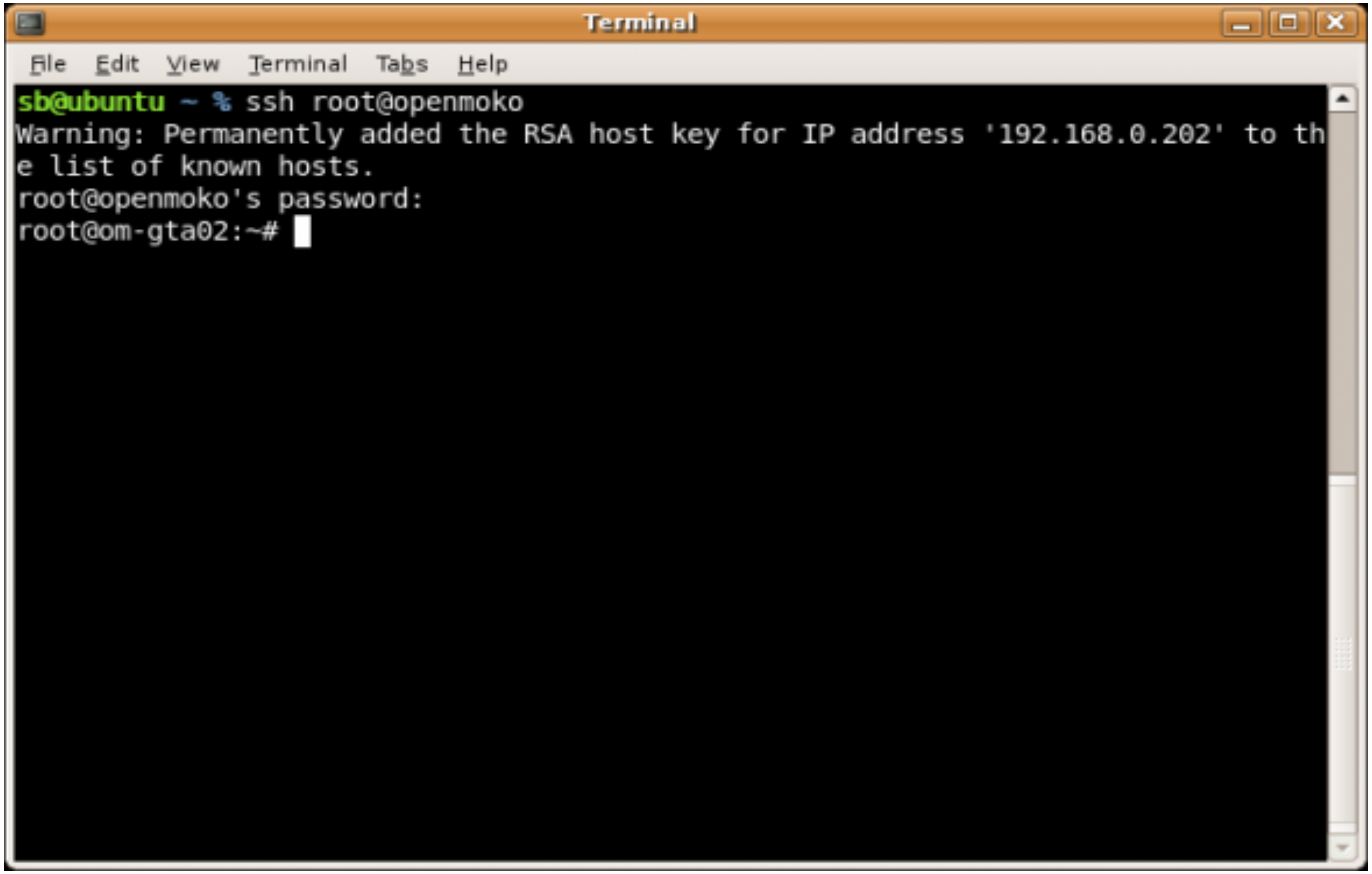
CGI is not persistent. It is meant to service requests, then to end execution.

We switched from a CGI architecture to a socket-based architecture.

select in GUI, operate in CLI

- User selects some files in GUI (e.g. some .tex and .eps files)
- User begins typing command in CLI (e.g. tar -czf archive.tgz)
- User drags files from GUI to CLI, and their names are inserted into the command line

Target User: Anyone who uses SSH!



```
sb@ubuntu ~ % ssh root@openmoko
Warning: Permanently added the RSA host key for IP address '192.168.0.202' to th
e list of known hosts.
root@openmoko's password:
root@om-gta02:~#
```

Timeline: Web Server

Sept. 24 - Nov. 5:
    Prototype execution of commands on SSH server, and getting input from client

Nov. 5 - Nov. 19:
    Integrate with frontend and test

Nov. 19 - Nov. 30
    Polish, improve efficency

Advantages

Can proxy through port 80 (for users behind a firewall).

One less reason to use X Windows.

Unmodified sshd.

Can easily browse without installing sshfs.

Disadvantages

- Easily encryptable with SSL, but server needs certificate.
- Cannot use public/private key authentication.
- Need an additional server (or daemon).

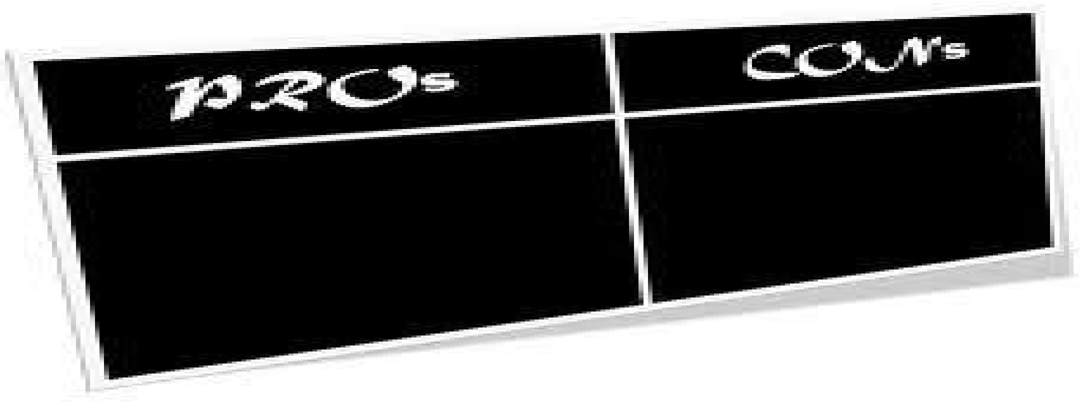
Analysis

We think that the simplicity of being able to have a graphical representation will outweigh these disadvantages.

CLI vs. GUI

For instance:

- CLIs are better for programmatic interactions and system administration tasks.
- But GUIs are much better file browsers.



Our Solution

We believe that they are best when used together.

Use Case 2:
File Browser

- Be able to browse remote file hierarchy.
- cwd of the terminal session follows the GUI.
- Perform common file manipulation operations in GUI
  - Delete
  - Rename
  - Copy
  - Cut

Use Case 3: Command Construction

Can select files in GUI and perform operation on them in shell.

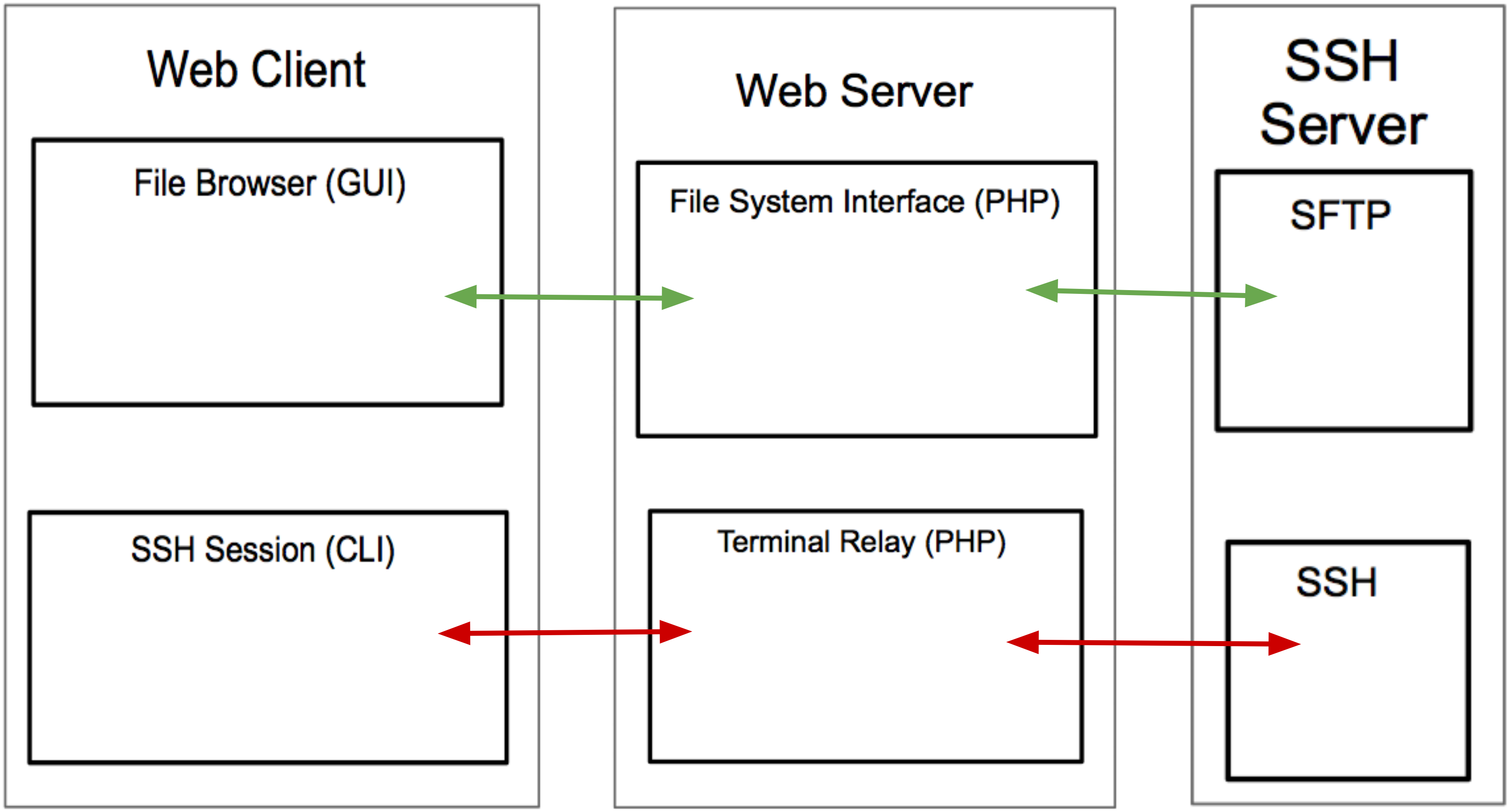
Drag-and-drop icons into a command.

File is converted to a string.

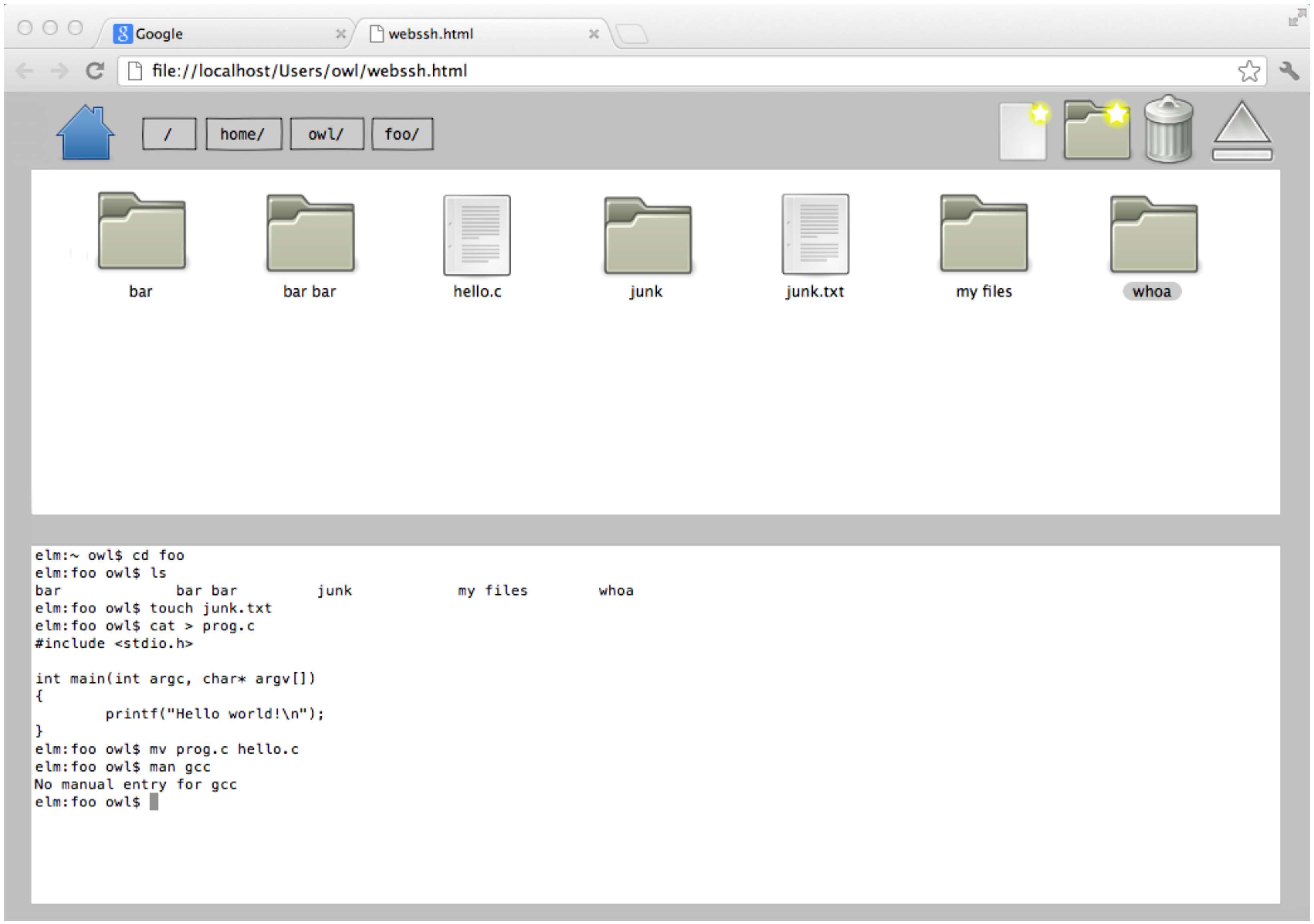
Use Case 4: Real Terminal Emulation

- (Nearly) any command which works over SSH should work over wSSH.
- Command-line text editors should work.
  - nano
  - vi
  - emacs

Architectural View: Logical



Mockup



```
elm:~ owl$ cd foo
elm:foo owl$ ls
bar             bar bar         junk            my files        whoa
elm:foo owl$ touch junk.txt
elm:foo owl$ cat > prog.c
#include <stdio.h>

int main(int argc, char* argv[])
{
        printf("Hello world!\n");
}
elm:foo owl$ mv prog.c hello.c
elm:foo owl$ man gcc
No manual entry for gcc
elm:foo owl$ 
```