WIDLE: A Web Based Linux Interface

Lucas Novoa, Charles Volzka, and Peter Bui
Department of Computer Science
University of Wisconsin - Eau Claire
Eau Claire, WI 54702
{novoalg, volzkacj, buipj}@uwec.edu

Abstract

For novice students coming from graphically oriented operating systems, navigating remote Unix command line shells can be as difficult as learning a new language. In order to reduce this complexity, we developed WIDLE, a Web based Interactive Development and Learning Environment. WIDLE is a RESTful web server that allows remote file system browsing, interaction, and editing through a standard web browser. It also provides an interface to view multimedia files from remote systems and an easy method to transfer files making it useful for advanced users as well as novices. In the following paper, we discuss our development methodology, the design of the project, and the implementation decisions we made. Furthermore, we provide an evaluation of WIDLE by presenting the survey responses from our first group of participants to test an alpha build of the project.

1 Introduction

Although Linux and other Unix-like operating systems are powerful programming environments, the command line interface of the standard Unix shell often serves as a discouraging barrier for novices. Coming from more traditional graphical environments such as Windows or Mac OS X, these students may view the Unix shell as intimidating and may possibly be deterred from continuing their exploration of computing due to its complexity. To address this problem, we developed **WIDLE**, a web server that supports basic file system navigation and file editing services in a graphical browser interface along with other advanced features such as the ability to view multimedia files, upload files, and receive notifications. By providing a convenient and approachable graphical platform for interacting with Linux systems, we hope to enable students to focus on high-level computing concepts instead battling the mechanics of Unix file system manipulation and operation.

The main contributions of this paper are the presentation of the design and implementation of WIDLE, and a discussion of the use of Agile development in managing this project.

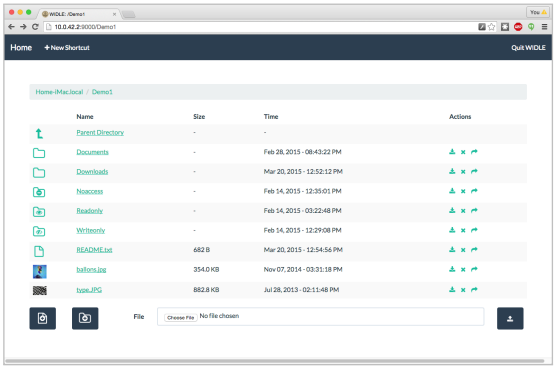


Figure 1: WIDLE - Folder View

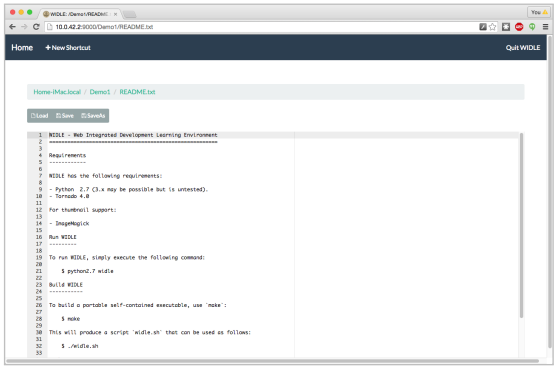


Figure 2: WIDLE - Editor

2 Background

The general idea behind WIDLE is inspired by the IPython Notebook [6], which is a web server that provides a convenient graphical interface for executing Python code on a remote machine via a web browser. In fact, many of WIDLE's features or characteristics are derived from such web-based applications. For instance, the procedure for starting WIDLE and authenticating with a secure token was again inspired by the IPython Notebook and the Cherokee [3] applications, which both use similar startup methods. Likewise, the notifications for user feedback are inspired by the website HackerRank [5]. Finally, WIDLE utilizes the popular Bootstrap framework [2] and Font Awesome icons [4], which are used in many modern web applications. What distinguishes WIDLE, then, is its focus on enabling novice uses to operate remote Linux systems via a graphical web interface.

The project's focus on students who are unfamiliar or uncomfortable with remote Linux system comes from the authors' experience. At many schools, systems programming

courses (typically involving C/C++) are taught using remote Linux terminals. Unfortunately, according to a recent report [10] in 2014, 94.67% of desktop web users used Windows or Mac OS X while Linux accounted for only 1.55%. While Windows and Mac OS X both offer command line tools, many users don't know of their existence and fewer still take advantage of them. Linux users are generally more command line savvy but make only a tiny faction of the user base. This means that for the majority of students, a systems class will likely be their first time working extensively in a command line environment. This creates extra cognitive load for students who we believe could be better served by focusing on new programming ideas and less on environment navigation. This is where **WIDLE** comes in.

WIDLE was born out of a desire to reduce this extra load without needing to use complicated integrated development environments such as Eclipse, Visual Studio or Xcode. By running a WIDLE server on the remote Linux system, the student will be able to easily navigate the file system in an intuitive graphical interface right from a web browser. In addition, we have included functions to create and edit text files and view multimedia files. We believe these features will make WIDLE a useful tool, not only for novices but also intermediate and advanced students. In addition to being useful, we hope WIDLE will give teachers options to create assignments with multimedia components in addition to traditional text manipulation and numerical computing.

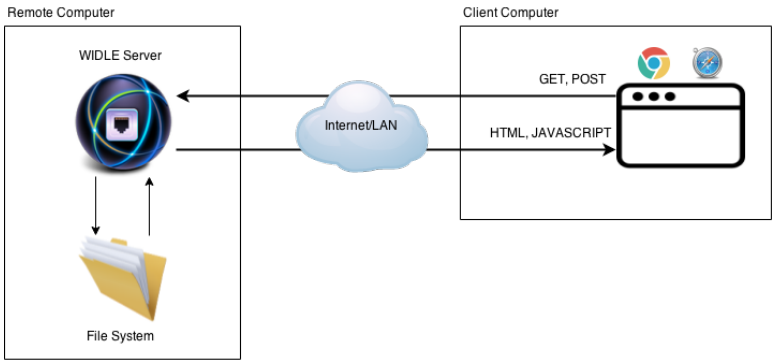


Figure 3: WIDLE - Design

3 Design

The overall design of WIDLE is shown in Figure 3. As shown in the diagram, the focus of our project is creating a web server which acts as a bridge between a remote Linux system and a local user. To ensure the correct file permissions are enforced, each user must first login to the remote Linux system and launch their own WIDLE instance. At launch, WIDLE displays a URL and a randomized session token to ensure secure access to the web interface. After logging in, the user sees a list of files from their home directory as shown in Figure 1. Files and folders are *clickable* with different actions based on type. For example, clicking on a folder will display its contents in the same folder view, while selecting text

files will lead to a page with an inline text editor as shown in Figure 2. Furthermore, files whose *MIME* types are recognized by the browser are displayed automatically by the web browser (i.e. images, audio, video etc.).

From this folder view, WIDLE supports a variety of common file manipulation operations. Items in the folder view can be deleted, moved, renamed, and new files and folders can be created. Some additional features include:

1. Shortcuts to quickly jump between folders
2. Upload files from the browser to the remote file system
3. Download files or entire folders in a single click. Folders are downloaded as a zip file with their original hierarchy preserved.

Operations are represented as simple icons with tool-tip style descriptions. Deviating from the Unix "no news is good news," we also incorporated a color coded notification system to inform the user of the success or failure of various actions as shown in Figure 4.

Figure 4: WIDLE - Success and Failure Notifications

4 Implementation

To implement WIDLE, we utilized a variety of web technologies. We used Python as our main programming language along with JQuery, Bootstrap and JavaScript to provide formatting and dynamic client side operations. We incorporated Tornado [7] as the base web server framework. In addition, the ACE [1] editor was integrated to provide text editing functionality and basic syntax highlighting. For reference, our implementation of WIDLE can be accessed at `https://bitbucket.org/pbui/widle`.

Most of the functionality of a WIDLE server comes from the implementation of custom handlers. Tornado maps these handlers to specific URL patterns that respond to HTTP `GET` and `POST` methods. These custom handlers allowed us to use a RESTful [9] design approach that gives functionality to the web application while enabling a modular design.

4.1 BaseHandler

As the name implies, the `BaseHandler` is an abstract class which provides the other handlers methods to render web pages, provide error messages, and get current user information. All of the following handlers are derived from this base class and utilize its methods for consistent page generation and error handling. Each individual handler, then, implements either a `GET` or `POST` method (sometimes both), which defines what actions should be taken when the corresponding HTTP request is made from the client.

4.2 AuthenticationHandler and SessionHandler

In order to add a measure of security, we implemented the `AuthenticationHandler` and `SessionHandler` classes. These two handlers work together to make sure the user who started the server and the one at the web browser is the same person. When the user starts the WIDLE server, he is shown the server address along with an automatically assigned port number for his specific WIDLE instance. A unique session key is also created as a form of authentication to guard against malicious access by a third party scanning the range of expected ports.

Although it may seem tedious, having each user run his or her own WIDLE server serves two purposes beyond security. First the server can run as a process of that user meaning their file permissions are preserved. Second, WIDLE does not need root or administrator access to start a server. That is, any user with permission to access the remote system can start their own WIDLE server.

Other WIDLE handlers, described below, have an `authenticated` property which is checked in order to prevent a malicious user from directly sending requests to the WIDLE instance without completing authentication. Since we currently do not support pausing a session, the SessionHandler's only function is to terminate an active session. Currently we do this by redirecting the user to a page notifying them that the server has been terminated and then end the server's main I/O loop. To make sure the notification page fully displays, we added a two second delay between the page redirect and loop termination.

4.3 NotificationHandler

Demonstrated in Figure 4, the `NotificationHandler` is simple, but critical to the user experience. The server application stores a list of notifications that can be created by any of the other handlers. The `NotifcationHandler`'s GET method then returns the list of notifications as `JSON` which can be displayed via JQuery to the user. The handler then clears the list of notifications to prevent duplicates from being show on a redirect. In addition to displaying notifications whenever a page is loaded, we added a method to call for them explicitly as well. This allows us to display error messages after POST events such as `Save` or `Save As` where redirection proved unsuitable because of the inadvertent loss of unsaved changes.

4.4 ShortcutHandler

The `ShortcutHandler` provides an extra level of convince for the user. This handler allows the user to easily save links to previously visited locations in the file system. When its POST method is called, the currently displayed path is added to the list of shortcuts. We used JQuery and AJAX to allow users to remove shortcuts by dragging them out of the shortcut bar. A POST is called with the path of the selected shortcut and an extra `'remove_shortcut'` argument. The list of shortcuts is then enumerated and the matching shortcut is removed from the list.

4.5 ThumbnailHandler

The `ThumbnailHandler` adds extra convenience by displaying thumbnails for image files in the folder view instead of the default file icon. When requesting the icon for image files, the folder view redirects the request to the `ThumbnailHandler`. First, the handler creates an SHA1 hash of the requested picture (a secure 160-bit hash value). If a thumbnail does not already exist for the file, a new one is created from the picture and placed in the cache folder named as `<hash>.jpg`. Finally the thumbnail for the image returned to the browser as the icon for the requested image.

We chose to name the thumbnails based on a hash of the image for three reasons. First, this allows us to store the images in a flat folder without worry of name collisions. Second, this method enables constant time lookup of the thumbnails regardless of the number of thumbnails stored. Lastly, if the same image existed in multiple locations, this approach would only need to create one thumbnail as multiple copies should hash to the same value.

4.6 FileHandler

The `FileHandler` provides the methods which let the user interact with the remote computer's file system. When displaying folder contents, we use stacked Font Awesome icons to indicate whether the items are files or folders. If an item has special permissions such as read only, write only, or no access, we stack an additional sub-icon to the item to visually indicate those permissions to the user.

The `FileHandler` uses a dictionary to match argument strings to `lambda` callback functions. Some filtering is applied based on the path the handler receives and whether the path is a folder or a file. Folders display their contents as shown in Figure 1. Files are filtered on mime types. Text files open in the ACE editor as shown in Figure 2, while multimedia files are viewed in browser. Finally remaining files are directly downloaded to the client's machine when clicked.

In addition to viewing files and folders, the `Actions` column provides an interface to the extra features of the `FileHandler`. Files and folders can be moved, deleted, or downloaded. The download action for folders required special implementation. First, we create a new temporary folder in the system's `temp` directory, this way we can name the zip file the same as the original folder without fear of name collision. Then, we walk the

path of the original source folder copying its contents to our new zip file. Thus, we walk the path item by item to make sure we preserve the hierarchy of the source in the new zip file. Finally, after the file is created, it is downloaded by the client and deleted from the `temp` directory. We also watch for exceptions to ensure the zip file is removed even if an unexpected error occurs.

4.7 Development Strategy

In addition to deciding on an implementation design we also needed to adopt a development strategy that allowed us to work as a team despite conflicting schedules. To accomplish this, we used a modified Agile development methodology. Recognizing the chaotic schedules of those involved, we followed Agile's "individuals and interactions over processes" [11] philosophy. We decided to integrate our scrum and sprint meetings into single weekly events augmented with chat sessions over IRC channels as needed. In order to keep the project manageable, sprints were fixed at one week.

As needed our weekly meetings would be used as longer project overview meetings where features and general project goals were decided. Following the Agile approach, the project evolved as it progressed. For instance, we moved from focusing on new students to also adding multimedia features for more experienced users based on feedback. Tasks would be broken down as much as necessary to turn them into one week tasks. In this manner, new features were added iteratively over time with the project left in operational states between each task. We used Trello [8] to maintain a backlog list of tasks and record discovered bugs. Reported bugs would be addressed as future tasks.

The more common, shorter weekly meetings focused on what we accomplished the week before, discussed what was and wasn't working in the process, and pulling from the backlog. As a team, we discussed our availability and skills to split off tasks rather than have them assigned by a managing authority. Additionally, the weekly meetings kept all of us in contact and up to date throughout the project. The fixed one week sprints provided a consistent momentum throughout development. It prevented individual tasks from being insurmountable while also ensuring continuous progress.

Overall, the use of the Agile development process allowed for the students to take equal ownership of the project and empowered them to not only realize their goals but to also shape the project's evolution and direction.

5 Evaluation

To evaluate WIDLE, we gave a class of 12 systems programming students a chance to use an alpha version of the project, and had them complete a survey of their experience. With the small sample size, the results should not be considered conclusive, but should still serve as an excellent feedback mechanism for this stage of the project. The survey included both quantitative and qualitative questions. To prepare the students, we started with a short demonstration on how to start a WIDLE server as well as an overview of its

features. After the demonstration we had the participants complete a series of self guided tasks (Figure A.1) designed to give them a chance to experience each of the features. After the self-guided tour, we asked the participants to fill out the survey of their experience. The results of the quantitative questions are shown in Figures 5 - 11.

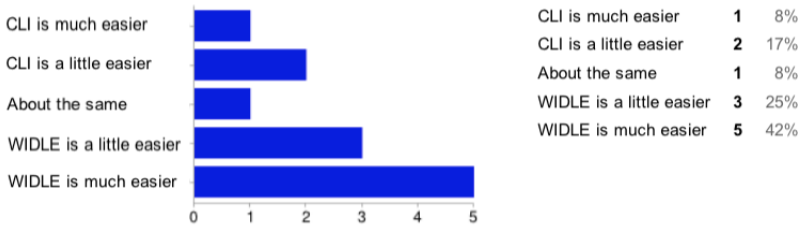


Figure 5: Results of first question. ”Initial learning curve, which is easier to use?”

5.1 Quantitative Questions

The quantitative questions focused on measuring participants reaction to WIDLE. Many of the results favorably skewed towards WIDLE over the command line interface despite its alpha standing. Figure 6 was closer to neutral and Figure 7 slightly favorited the command line interface. Further study is needed but for Figure 6, a common sentiment during the demo was a desire for tab completion when interacting with directories. This feature was previously planned for inclusion in the project and it will be interesting to compare results after it has been implemented.

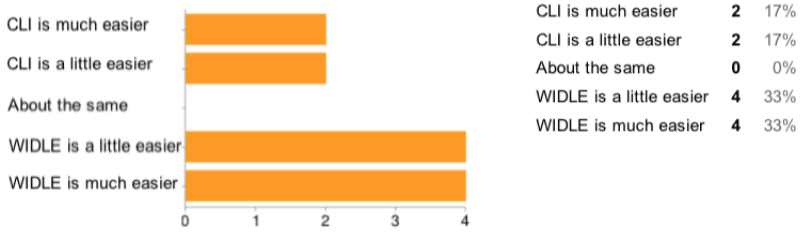


Figure 6: Results of second question. ”Navigation (changing directories, viewing files), which is easier to use?”

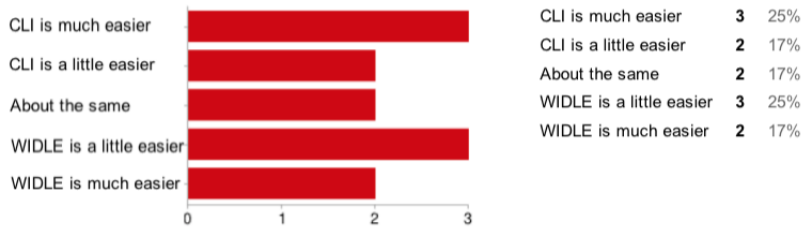


Figure 7: Results of third question. ”Opening/Editing documents, which is easier to use?”

The slight favoring of the command line interface in Figure 7 is more confusing. In this alpha build, the ACE editor's capabilities are limited and similar to the command line's `nano` program. Since the participants have already completed a half semester of C programming, it is possible they have already become familiar with more advanced features of editors like Vim or Emacs. Alternately the alpha version lacks the ability to execute edited source code which may also effect utility of editing files from the browser.

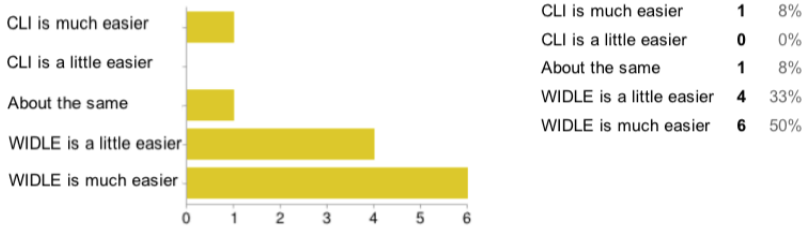


Figure 8: Results of fourth question. "Viewing multimedia files, which is easier to use?"

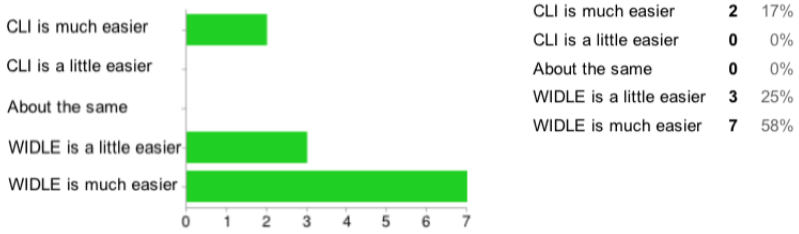
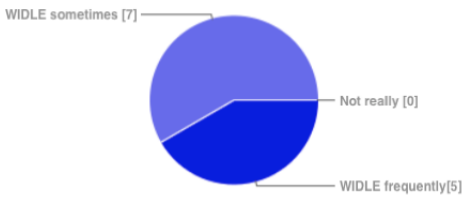


Figure 9: Results of fifth question. "Transferring files from the local computer to remote systems, which is easier to use?"

Figures 8 and 9 showed a strong favorable response towards WIDLE. While the project's primary goal as an entry level tool for new students responded well, its secondary goal of making remote multimedia and file access easier for advanced users appears to have had a stronger response. Once the project reaches a beta stage, another demonstration at the start of the semester may show a greater balance. Regardless, we believe it is clear that development should continue enhancing these features and place them in equal priority with the original WIDLE vision.

Figure 10: Results of sixth question. "Would WIDLE have made your introduction to systems (C) programming easier?"

Finally, Figures 10 and 11 indicate that even after a brief demonstration of WIDLE, the students felt that the application was both useful and would make their systems programming class easier.



WIDLE could frequently come in handy	5	42%
WIDLE could sometimes be useful	7	58%
Not really	0	0%

Figure 11: Results of seventh question. ”Do you think WIDLE would be useful as a general tool for connecting to remote Linux systems?”

5.2 Qualitative Questions

visibility
The ability to quickly zip and download files.
Tab Completion
It's nice to be able to view multimedia so easily.
Shortcut creation
The uploading seemed the most useful. A sftp connection isn't hard to make but this definitely made the process easier.
The ability to view videos and images in the browser in chrome, and the upload features were nice.

Figure 12: Results of the eighth question. ”What feature(s) seemed most useful?”

For the qualitative section of the of the survey we asked participants four questions related to their experience with WIDLE. Unlike the quantitative portion of the survey, these questions did not require any answer. Figure 12 showed participants liked a wide variety of features of the project. While viewing multimedia was mentioned twice, many features were mentioned by respondents as useful. A notable absence is any mention of the editing features. Given Figure 7 this was not surprising and means either extensive work is needed on the utility of this feature or development resources should be diverted to other features.

Tab completion A directory tree as the background The ability to still type commands
I would like to see a terminal function added.
drag + drop for file uploads as an alternative to the browse dialogue.
Run commands from the web page.
It would be nice if there was a way to automatically start a web browser session instead of having to copy and paste the codes into a web browser. That part is the most tedious part of using the system.

Figure 13: Results of the ninth question. ”What feature would you most like to see added?”

The question in Figure 13 was designed to give us feedback on where to focus future development on the project. From the comments it appears there are two general requests for future development. First, there is a strong sentiment to bring more terminal features to the project. This includes adding tab completion to the dialog boxes as well as an ability to issue arbitrary terminal commands from the web page. Second, there is a desire to increase

the ease of use for the project as a whole. Tab completion again falls under this category as well as removing the need to start a server from a `ssh` connection.

Wilde would not run mp4 files.
-
In chrome - new file, new dir, and upload buttons get a little weird when browser is not full screen - just barely noticed.
not yet
Deleting the scratch folder because of a lack of permissions. Not really a bug, just maybe should fix for the self guided demo.
The only bug I found (that wasn't already known) was trying to upload a video that was quite large (140mb I believe)
Nope, not that I saw.

Figure 14: Results of the ninth question. ”Find any bugs? What were they and how critical”

The few noted bugs in Figure 14 focused on issues with multimedia files. It showed that we need to expand breadth of testing for such files. Our previous method of testing a few random files did not catch issues such as videos that used a variety of encoding methods as well as testing the larger file sizes created by high-definition videos. This problem will only become more pronounced as we extend our list of supported browsers. It may be advisable that we incorporate a third party player, or formalize our testing procedure to catch such issues in the future.

Make the windows for new file, move, etc. a little bigger - I couldn't see the whole filepath, which was annoying. If this is meant for people who don't know how to use the command line, maybe make some of the commands a little more intuitive. (ex// Since I've used the command line, I know that rename = move, but someone who hasn't used it might not immediately make that connection.) Also, I don't know if there's anything you can do about it, but for non-users, the whole ctrl^C vs. right-click thing could take a long time to figure out. The shortcuts are very helpful, and the icons look very nice.
Good work fellas
It would be cool to be able to use typed commands at the same time. I still like the CLI, and feel like WIDLE is best-used as a teaching tool to help people transition to CLI, and-or as a CLI accessory to use at the same time for opening files that aren't made for the CLI.
I feel like this can be handy, but I also feel like learning CLI is fairly daunting. Having an option to be able to get around that might actually make "jumping in the pool" more difficult. Overall, however the software functions well and does what it attempts to do for the most part!
Sounds like it has promise. I would be interested in helping with the project or hearing more about updates.

Figure 15: Results of the ninth question. ”Any other thoughts, comments, or suggestions”

The final survey question in Figure 15 shows an overall positive response to the project. In addition to getting feedback about its current state, the demo has given us an opportunity to expose other students to the project. This positive exposure gives us an opportunity to demonstrate the utility of student research to younger students as well as the possibility of recruiting more student researchers to replace graduating members.

6 Conclusion

Our main goal in this project is to design a new interface that can ease the transition for new systems programming students and offer useful functionality for expert users. From Figure 10 it is clear more work is needed to complete our primary goal as an introductory systems tool. While 50% of participants thought it would have made their initial introduction to systems programming easier, the other 50% were ambivalent. That said, Figure 11 showed all the students believed WIDLE could be a useful general tool. Additionally their comments following the survey expressed interest in the future of the project, indicating that the project is on the right track.

We have also learned new things about software development techniques and early life cycle development. By applying an Agile style of development we were able to keep connected despite working individually in a distributed manner. Additionally by accepting feedback throughout the process we were able to redirect development time toward the multimedia functions of the project. While multimedia was not a key component of the original product vision, the flexibility allowed us to add features which ultimately had the strongest favorable response from the participants.

By presenting common file system tasks in a graphical browser interface, we hope new students can focus more on high-level computing concepts. New and experienced users alike should be able to appreciate the ability to easily view audio/visual files from remote Linux systems and easily transfer files without mounting remote systems or use of obscure command lines.

References

[1] Ace. ”Ace - The High Performance Code Editor For the Web” [Online]. http://ace.c9.io/. [Accessed: 28 Feb 2015].

[2] Bootstrap. ”Bootstrap: The World’s most popular mobile-first and responsive front-end framework” [Online]. http://getbootstrap.com/. [Accessed: 28 Feb 2015].

[3] Cherokee. ”Cherokee Web Server” [Online]. http://cherokee-project.com/. [Accessed: 28 Feb 2015].

[4] Font Awesome. ”Font Awesome, the iconic font and css toolkit” [Online]. http://fontawesome.io/. [Accessed: 28 Feb 2015].

[5] Hacker. ”Programming problems and challenges :: HackerRank” [Online]. https://www.hackerrank.com/. [Accessed: 28 Feb 2015].

[6] IPython. ”IPython: Interactive Computing”. [Online]. http://ipython.org/. [Accessed: 28 Feb 2015].

[7] Tornado. ”Tornado Web Server”. [Online]. http://www.tornadoweb.org/en/stable/. [Accessed: 28 Feb 2015].

[8] Trello. ”Trello”. [Online]. https://trello.com/. [Accessed: 28 Feb 2015].

[9] C. Pautasso, O. Zimmermann, and F. Leymann. ”Restful web services vs. ”big” web services: Making the right architectural decision”. In Proceedings of the 17th International Conference on World Wide Web, WWW 08, pages 805814, New York, NY, USA, 2008. ACM.

[10] ”Desktop Operating System Market Share”. Jan 2015. [Online]. Netmarketshate http://marketshare.hitslink.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0&qpsp=2014&qpnp=1&qptimeframe=Y [Accessed: 28 Feb 2015].

[11] Beck. K. et al. ”Manifesto for Agile Software Development”. 17 Feb 2001. [Online]. http://agilemanifesto.org [Accessed: 28 Feb 2015].

Appendix

WIDLE - Self Guided Demo

To Start

1. ssh into dplsubmit
2. run command: /data/software/all/bin/./widle
3. Copy and paste* url into browser
4. Copy and paste* session token in to WIDLE
   *Remember in Putty to copy text, select it and right-click (or use Putty menu)
   *In Putty control-c quits the server instead of copying the text

1. Shortcuts
   1. Navigate to /data/scratch/widle/<uwecid>
   2. Create a shortcut
   3. Click “Home”
   4. Use shortcut to return to your scratch folder

2. Moving files
   1. Rename colored pictures as red.jpg, blue.jpg, green.jpg
   2. Create a folder called “b and w”
   3. Move the black and white pictures to the new folder

3. Editing
   1. Open the source code and make some changes

4. Downloading Folders
   1. Go up one folder (Parent Directory)
   2. Download your scratch folder
   3. Expand zip file on the local machine. F

5. Deleting
   1. Delete your scratch folder from dplsubmit
   2. Try the shortcut you created earlier
   3. Remove the shortcut to your scratch folder

6. Uploading
   1. In another tab/window, go to http://www.videezy.com
   2. Pick a short clip and download it
   3. Go to your home folder and upload the clip
   4. View the movie from dplsubmit in the browser

7. Quit WIDLE
   1. Or leave it running if you like
   2. Fill out survey at: http://bit.ly/1MrVIss

Figure A.1: Self guided demo given to survey participants