uAssign: Scalable Interactive Activities for Teaching the Unix Terminal

Jacob Bailey
University of Illinois at Urbana-Champaign
Urbana, Illinois
jbbaile2@illinois.edu

Craig Zilles
University of Illinois at Urbana-Champaign
Urbana, Illinois
zilles@illinois.edu

ABSTRACT

We describe uAssign, an assignment system for teaching and assessing command line terminal skills. uAssign allows instructors to create auto-graded terminal assignments that require students to perform a high-level action that can be completed in many ways. Assignments can be randomized so that students can't re-use old solutions. uAssign is implemented via an in-browser terminal emulator that uses WebSockets to connect to a Docker container. Performance testing and its use in a large-enrollment lecture course show that it is efficient enough to handle a large number of concurrent users. A survey of students shows significant improvement in terminal skill confidence after using uAssign and that students have a high level of satisfaction with uAssign assignments.

CCS CONCEPTS

• **Applied computing** → **Education**; **Computer-assisted instruction**; *E-learning*;

KEYWORDS

assessment, Unix, terminal, command-line, auto-grading, scalable

ACM Reference Format:
Jacob Bailey and Craig Zilles. 2019. uAssign: Scalable Interactive Activities for Teaching the Unix Terminal. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), February 27-March 2, 2019, Minneapolis, MN, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3287324.3287458

1 INTRODUCTION

The continued increase in enrollment in computer science programs has created demand for scalable assessment technology for homework and exams. Technology for code auto-grading and plagiarism detection are widely used, and recent research has explored scalable techniques for giving personalized feedback on style and design [10, 13]. This technology, however, has largely focused on programming assignments, which is but only one part of being a successful computing professional.

One of the areas that has not been given a lot of focus is *terminal skills*. The terminal, or command-line, generally refers to the text-based interface that exists on nearly all computers. At a terminal, you have access to common utilities to perform actions on the system (file management, program execution) without using a graphical interface. A user can perform simple tasks, but also construct more complicated commands to do things that would be inefficient to do by hand. These sorts of skills are an important tool in the toolbox of computing practitioners but are often relegated to course Wiki pages and informal help from lab instructors, with assessment either non-existent or only through basic testing with multiple-choice quizzes.

In this paper, we describe **uAssign** (available as open source [5]), a scalable approach to teaching and assessing Unix terminal skills. In designing uAssign, we had four main goals:

(1) The assignments should provide an authentic terminal experience where students can explore the current state of the machine and complete the assignments using many possible strategies (i.e., not merely checking that the student typed in a specific command).
(2) The platform should be flexible enough to support a wide range of terminal assignments and, also, support randomization so that unique problem instances can be given to each student to prevent sharing solutions.
(3) The platform should have high availability and scale gracefully, preventing abuse and resource exhaustion independent of user actions.
(4) Terminal assignments should be accessible to students in a convenient, platform-independent manner, ideally available through the the same interface that students are using for their other online homework.

uAssign achieves these goals through the composition of a collection of existing technologies, including Docker containers, WebSockets, and an in-browser terminal emulator. Using uAssign, a student will be able to open their browser and be presented with a real, working terminal attached to a live system in which they can complete their assignment, as shown in Figure 1.

Specifically, this paper makes the following contributions:

(1) We describe the design of the uAssign server, including its management of container life cycles and its integration into a representative web homework interface in Section 2, with special attention to how to quickly provision containers in Section 2.1 and security in Section 2.2.
(2) We present the results of a survey including information on students' prior terminal experience and the efficacy of uAssign in teaching terminal skills.
(3) We present performance results demonstrating that uAssign scales sufficiently for large enrollment classes.

SIGCSE '19, February 27-March 2, 2019, Minneapolis, MN, USA

ACM ISBN 978-1-4503-5890-3/19/02.
https://doi.org/10.1145/3287324.3287458

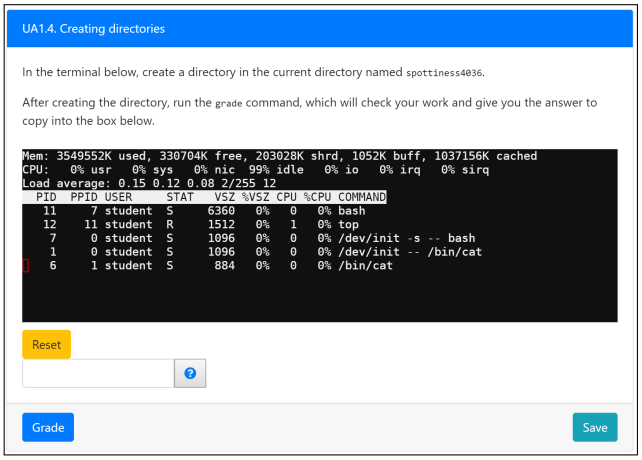

Figure 1: An example uAssign session for an introductory question demonstrating that the terminal has standard capabilities like `top`.

Additional implementation details and survey results can be found in Bailey's Master's thesis [6].

2 DESIGN AND IMPLEMENTATION

The key idea for making terminal assignments that provide an authentic terminal experience is to provide students access to an actual terminal that they can interact with. The uAssign system does this through running a Docker[1] container (a light weight "virtual machine") for each student assignment on a server and providing access to these containers through an in-browser terminal emulator. This requires students to be on the Internet when they complete the assignment, but provides low latency interaction with a terminal in a secure, platform independent way.

In principle, uAssign's software architecture could be paired with any learning management system (LMS); the implementation of uAssign that we describe below is for an embedding in the PrairieLearn [18] LMS that is used for other assignments in the course in which it was deployed.

The uAssign terminal assignment system is composed of three independent parts, illustrated in Figure 2:

(1) The uAssign server (shown on the right), responsible for managing the lifetimes of assignment instances and their associated Docker containers on the server,
(2) A PrairieLearn server-side component (bottom left), responsible for providing assignment specifications (randomly parameterized to prevent cheating) and recording completion in the grade book, and
(3) A PrairieLearn client-side terminal interface (top left) that mediates the interaction between the user and their container.

The standard flow through the system begins with a request for a new problem, causing the server-side PrairieLearn component to

[1]Docker is a popular container platform which makes use of a kernel's isolation features [1]. Core to Docker is its daemon (`dockerd`), which manages containers and exposes a REST API, allowing a client to build images, create and manage containers, as well as other container-related tasks. This library-like interface enables Docker to be used as part of a system that wants to work with containers programmatically.

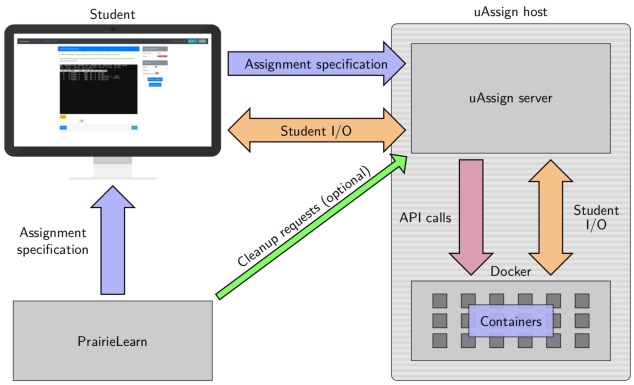

Figure 2: uAssign comprises three actors: (1) the LMS (PrairieLearn) provides assignment specifications, (2) the student's web browser displays the terminal, and (3) the uAssign server manages the Docker containers and mediates communication between the student and their container.

construct an assignment specification. This specification consists of a description of how the uAssign server should initialize the Docker container, including what files and directories should be created and any commands to be issued on the machine before control is transferred over to the user. To make assignments unique between users, we vary the names and contents of files and the names and hierarchy of directories. Our implementation encodes the specification as a JSON document with a UUID as a unique identifier and provides the specification to the student's web browser along with the client-side code.

The browser forwards this assignment specification to the uAssign server. Upon receipt, uAssign uses the provided UUID to see if this assignment is already in progress, and, if not, creates the container using the specification (discussed in detail in Section 2.1) and an *instance* record to track the relationship between this assignment specification and the container. If the instance record already exists, perhaps because the student is returning from having navigated away from the assignment, they are re-connected to their container.

The terminal session is then displayed in the student's browser using a series of components. On the client side, we use the `hterm` [2] terminal emulator written in JavaScript and a WebSocket to communicate to the uAssign server. Since `hterm` only provides functions to write data to the terminal and hooks to capture input text and key combinations, as well as the terminal's dimensions, we had to write client side code to proxy data from `hterm` over the WebSocket. The protocol we use to communicate between the client and the server is one derived from the protocol used by terminado [9], an application for proxying a terminal to a Python Tornado server. The uAssign server then proxies the I/O to the container, using the instance records to identify the correct container.

Currently, grading is accomplished through the use of "secrets". This is a design decision resulting from a desire to loosely couple the uAssign server from the LMS that is hosting the assignments. In some cases, the secret is discovered by doing the assignment (e.g., a `grep` assignment asks students to provide the name of the file containing a given string). In other assignments, a `grade` executable is placed in the container, which students can run to validate that a task (e.g., the appropriate set of files were deleted) was done

correctly and dispense the secret. The secrets are unique for each student and provided to the container as part of the assignment specification. Students cut and paste the secrets from the terminal window into the encompassing PrairieLearn window and press the "Grade" button, then presented with their score.

When an assignment is graded, PrairieLearn sends the uAssign server a "cleanup request", which instructs the uAssign server to cleanup any resources associated with that specific assignment specification. These cleanup requests are provided on a best-effort basis. The uAssign server will also deallocate resources for an assignment if it has been idle for a few hours.

The user interface also includes a "reset" button. When pressed, the client will instruct the uAssign it to force an early cleanup of the active instance. Once the instance has been removed, the client will repeat sending its assignment specification. Since the instance was removed, the student will be presented with another instance of the same specification, effectively resetting the assignment.

2.1 Building containers faster than Dockerfiles

uAssign does not use Docker's standard method of configuring containers (`Dockerfiles`) because they are much too slow for our purpose. Our use of Docker is very different from what its authors intended. Docker was designed with the expectation of simultaneously running a large number of containers, but generally initializing those containers using a relatively small number of "images". As such, Docker does not optimize generating new images for speed, but, rather, for minimizing storage and network bandwidth. Docker images are internally organized into layers, where each layer represents the difference between the current image and the last, permitting significant sharing between similar images.

In uAssign, however, we want every container to have a unique image, so that we can give slightly different versions of the assignment to each student. In early versions, uAssign generated images with templatized Dockerfiles, providing the specification as the template's context. But, Docker's excessive I/O from generating the layers – each directive in a Dockerfile adds a layer – created problems both from a latency standpoint and caused I/O to become a scalability bottleneck. Also, Dockerfiles' use of syntax-significant whitespace made templating fragile.

Instead, uAssign skips this build process altogether and uses a custom JavaScript routine to perform actions directly on the target container as directed by the assignment specification. In this way, build time is reduced significantly. Our specifications even support identifying which actions can be performed in parallel, whereas Dockerfiles can only be executed one directive at a time.

We find that uAssign's implementation results in a 2 to 3 times speed up relative to a traditional Docker build. Figure 3 compares the container build time (averaged across 50 runs), as measured on the uAssign server, between our custom script (`index.js`) and the straightforward Dockerfile implementation. Across a collection of representative assignments, we find that uAssign's implementation is able to keep wait times in a tolerable range [15].

2.2 Security concerns and misuse protection

uAssign was designed to be as secure as possible while still providing students with an accurate environment to complete their

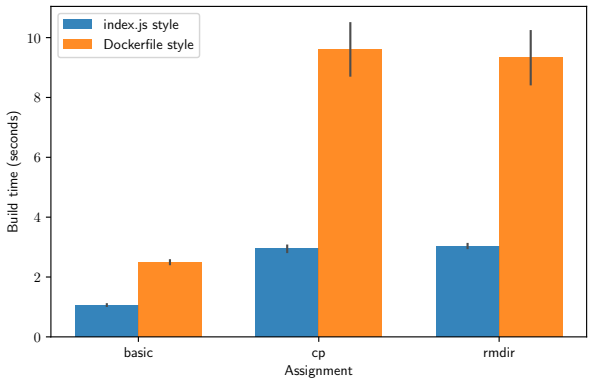

Figure 3: A comparison of build times using uAssign's actions (`index.js`) and Docker's own build system. The "basic" assignment asks students to read a file and answer with its contents. The "cp" assignment asks students to copy a file from one location to another. The "rmdir" assignment asks students to remove a directory. The "cp" and "rmdir" assignments have longer build times because they involve building a Go binary for grading.

assignments. Since uAssign gives a user access to a real container on a real machine, special care must be taken to ensure that privilege is not abused. uAssign, much like every other service which operates on the open Internet, needs to protect against general third-party attacks. As such, it implements standard security measures, including HTTPS (with built-in LetsEncrypt support for free certificate issuance), Secure WebSockets, and a CORS policy.

But, in addition to those third-party attacks, uAssign also needs to be able to protect against potential attacks by its users on the assignments themselves, as well as the resources that the assignments use. There are two main concerns: (1) students attempting to get credit without completing the assignment, and (2) users using the containers for something other than their intended purpose.

The most obvious point of attack for bypassing the assignment is the assignment specification, which contains all of the information needed to solve the problem, including the secret itself. Perhaps surprisingly, our implementation of uAssign actually passes the specification through the client browser in its effort to decouple the uAssign server from PrairieLearn. To prevent the user from accessing the contents of the specification, it is encrypted using shared-key encryption via AES with HMAC-SHA-256 verification.

Additionally, during the development of the course material for uAssign, we found that some students were able to extract strings from our grading binaries (written in Go), giving them a very easy way to obtain secrets. In order to mitigate this sort of attack, the secrets are compiled in an obfuscated form (gzip'ed, then XORed with a mask, and then base64 encoded), which the grading binary can decode when it runs. While this method is not foolproof, it would require more effort to bypass than to just do the assignment.

Because uAssign provides command line access to the containers it hosts, we must make efforts to prevent the containers themselves from being abused (e.g., to run Bitcoin miners or DDOS attacks). Our container security has three main elements: (1) Docker-imposed

resource limits, (2) disconnecting the container from the network, and (3) preventing simultaneous connections to a container.

Docker natively provides the following resource limiting features [3]:

- Memory usage limits, by specifying a hard maximum in bytes.
- CPU quotas, by specifying what percentage of the system a container is allowed to use before being unscheduled.
- Process limits (similar to `ulimit`), specifying the maximum number of processes a container can spawn.

The main resource that isn't well limited is disk storage. Docker *has* the ability to limit storage usage, but only via a global setting. For example, setting the limit to 500 MB means that a container which reaches 500 MB of data will no longer be able to write to its filesystem. This cannot be customized per container, so the limit must be set to the largest size they expect *any* container to use.

Docker also manages networking for its containers, creating networks and virtual network adapters. In the case of uAssign, networking is available to a container while it is being configured, but is disabled before a student has access.

Finally, instances are only allowed to be connected to a single terminal emulator at a time. If another client attempts to connect to the same instance, the first connection is terminated, and the second takes its place. This prevents uAssign from being used for unintended purposes that involve multiple connections to the same container, such as a make-shift chat room.

3 EXPERIMENTAL RESULTS

During the Fall 2017 and Spring 2018 semesters, uAssign was used in a second programming class for CS majors. In was used by 127 and 264 students, respectively, in the Fall and Spring semesters. The students were given five weekly homework assignments containing 35 exercises that included directory navigation, listing, and removing, file copying, moving, renaming, and deleting, hidden files, file modification times, globbing, grep, tar and zip files, find, pipes, and C compilation. Students on average take about 3 minutes to do each exercise.

We conducted a survey during the Spring semester covering student preferences of operating system, their confidence in terminal skills, and uAssign's usability, which is described in Section 3.1. In Section 3.2, we describe an in-class stress test that we performed to identify scalability bottlenecks.

3.1 Student Survey Results

With IRB approval, the survey was performed online and students were incentivized to participate in the survey via a raffle of gift cards. In total, 57 students participated out of an enrollment of 264, constituting a respondent rate of 22%.

We asked students to self-rate their proficiency in the terminal on a scale from 1 to 5 (from "no experience" to "very experienced"), both before and after completing uAssign's terminal assignments, as shown in Figure 4. Over all respondents, we saw the mean increase from 2.40 to 3.42 – an increase of 1.02 on a five point scale – from completing the assignments. Breaking down the data by operating system preference (Table 1), Windows users self-report benefiting the most from the terminal assignments, followed by MacOS users.

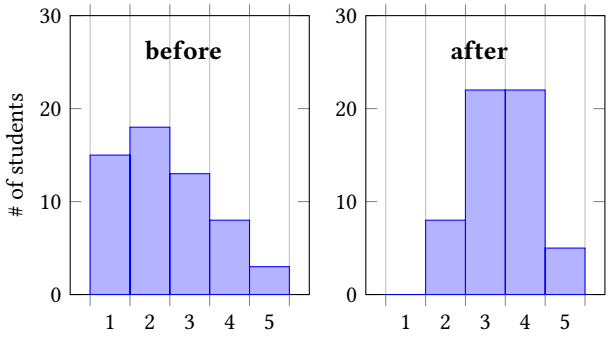

Figure 4: Student responses to "Rate your proficiency with the command line" before and after the terminal assignments.

Users who used Linux as their primary operating system reported no improvement, likely because they were already proficient.

We also see positive results with respect to the affective dimension. Students have very high perceptions of the utility of terminal skills (mean: 4.67, std dev: .58), as shown in Figure 5. Students were likewise satisfied with uAssign's user interface (mean: 4.37, std dev: .67, Figure 6 Perceptions of whether the assignments were good use of their time were also positive, but not as strong (mean: 3.63, std dev: .99, Figure 7).

Our survey suggests, however, that our assignments have likely not yet given students enough practice with the terminal. Only 60% of the participants stated that they felt confident they could use common terminal utilities, 40% still either disagree or are unsure (mean: 3.75, std dev: 1.11, Figure 8). When asked about their confidence with reading `--help` and `man` pages, 35% of participants aren't confident (mean: 3.86, std dev: .97, Figure 9). Also, participants still tend to prefer GUI tools to the command line for filesystem tasks (mean: 3.46, std dev: 1.18, Figure 10).

3.2 Performance stress testing

The only heavyweight activity on the uAssign server is spinning up new containers; once the container exists, it takes minimal network bandwidth and CPU to handle the terminal interactions. Spinning up containers became a significant bottleneck in our original Dockerfile-based implementation in Fall 2017, with students having to wait multiple minutes for their terminal to appear when the server was loaded.

With the revised container-building implementation described in Section 2.1, we stopped receiving performance complaints from students in Spring 2018. While this was a clear improvement, we weren't confident that we had removed the bottleneck, since students typically complete assignments on their own schedule over the course of many days.

To observe the server's behavior under controlled load, we performed a pair of stress tests in Spring 2018, where students were asked to access their terminal assignments simultaneously in lecture. Each lecture section involved roughly 80 concurrent users completing assignments with the uAssign server running on an AWS `t2.medium` instance. In both tests, mean and median container construction latency increased to around 15 seconds, which

Table 1: Self-reported improvement in terminal skill confidence, by operating system.

Primary		Primary + Secondary	
OS (% of participants)	Avg. Improvement	Secondary OS	Avg. Improvement
Windows (54.39%)	+1.323	Linux (15.79%)	+1.889
		N/A (36.84%)	+1.095
		MacOS (1.75%)	+1.0
MacOS (40.35%)	+0.739	N/A (28.07%)	+0.875
		Windows (3.51%)	+0.5
		Linux (8.77%)	+0.4
Linux (5.26%)	+0.0	Windows (5.26%)	+0.0

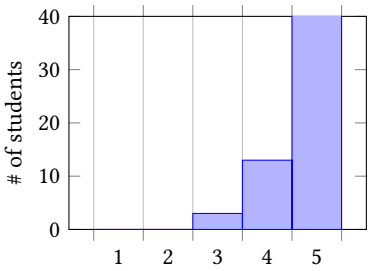

Figure 5: "I believe terminal skills are useful."

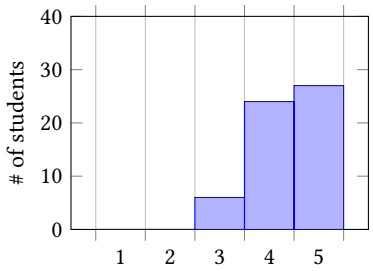

Figure 6: "The terminal assignment system provided a friendly user interface."

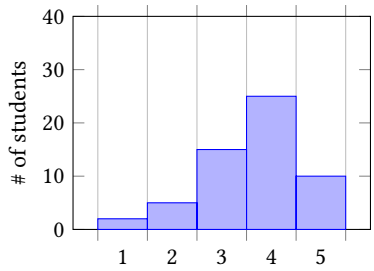

Figure 7: "My time was well-spent completing terminal assignments."

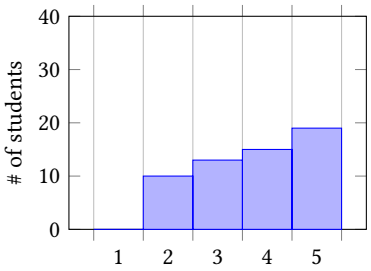

Figure 8: "I am confident that I can use common terminal utilities."

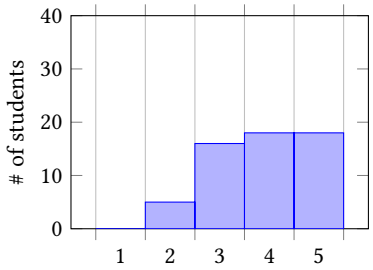

Figure 9: "I am confident that I can read terminal utility 'help' documentation (`-h`, `--help`) and man pages."

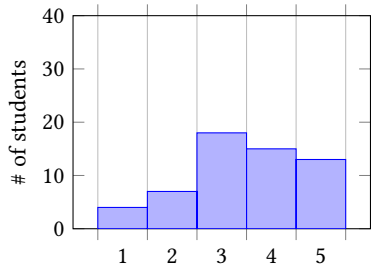

Figure 10: "I prefer performing common filesystem tasks in a GUI, rather than in the terminal."

is tolerable, and the terminals had no noticeable latency once their containers had been created.

These results make us comfortable running assignments with uAssign for class sizes up to 500 students on `t2.medium` instance, because the rate that students will construct containers is practically limited by the rate at which they can complete the assignments, even if all of the students were to wait to complete the assignments until the deadline. We believe that a slightly larger AWS instance with additional CPU and I/O bandwidth would be sufficient for even the largest computing classes at our university. Data collected on the uAssign host using `collectl` [17] shows that only CPU utilization was significantly affected by the stress test as shown in Figure 11; memory usage increased only a few hundred megabytes, and network usage was negligible. Additionally, scalability could be increased through the use of a filesystem with native Docker storage support like `btrfs`, where no extra work is needed to create a container's storage space.

4 RELATED WORK

Previous efforts to support the teaching and assessment of terminal skills can be categorized into four main types: (1) self-guided instruction tools, (2) traditional lessons with pencil/paper quizzes, (3) the use of virtual machines to teach system administration, and (4) web-based systems that present tasks for students with a means for validating that the assignment was completed.

Included with UNIX v7, *learn* [12] created an interactive environment to teach terminal usage by interpreting "CAI" (computer aided instruction) scripts. Lessons varied from basic command usage and filesystem navigation to the C programming language. Intended for self-guided instruction only, this software lacked any sort of assessment outside of completion of a given lesson.

Terminal skills have also been taught using traditional in-class lessons and hand-graded quizzes [11, 16]. Using quizzes (be they paper or online) may not fully assess skills, relying on a student to answer a question by statement, rather than perform some task.

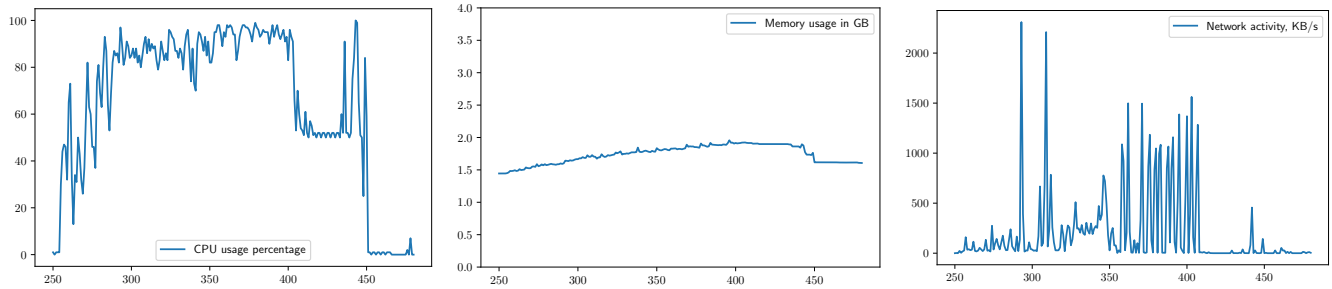

Figure 11: CPU, memory, and network bandwidth usage during our first stress test. The X-axis indicates time in seconds, with the stress test starting approximately 250 seconds after `collectl` was started.

Virtual machines are often used to teach system administration, so that learners don't break actual systems. "User Mode Linux" (UML) [8] allows students to connect to a custom Linux derivative running as a VM, over emulated serial lines provided by the VM host, where students can perform administration tasks such as simulated system updates and service management. "Virtual Lab" uses a web interface to provide access to its virtualized environment [7]. The client browser uses AJAX requests to send a user's command to the server. The server then uses SSH to run the command and captures and returns the full output back to the user as an HTTP response. This interface is not a "true" terminal, as it is not interactive (i.e., you cannot run editors, live-updating programs like `top`, or cancel execution). Neither system appears to integrate instant feedback/auto-grading.

The final category of related work are the peers of uAssign, web-based terminal assessment tools that permit randomization of assignments and support for auto-grading. "Unix-training" presents its lessons as a treasure hunt, where each step (e.g., compiling code, creating shell pipelines, accessing remote systems over SSH) gives you information to help you solve the next step [14]. Each student's path is unique to them, generated psuedorandomly (using some hashing of user information), and interactions with a web app tracks a student's progress. Unlike uAssign, this system relies on students having their own *NIX machines for course work.

The closest related work to uAssign is the concurrently developed TuxLab project [4]. Both uAssign and TuxLab work off of the same core principle – exposing customized Docker containers to students to complete terminal assignments. Both use instructor-written JavaScript code to efficiently configure containers by directly performing tasks on the container before the student connects. Both set resource limitations on the containers for performance isolation and to prevent abuse; both set CPU/memory limits, but uAssign's current deployment also limits disk usage.

The main difference is that TuxLab is designed as a stand-alone learning web app, where uAssign is designed to be integrated into an existing LMS. This design choice gives TuxLab an extra level of control and a more integrated user interface (a terminal on the right, reference on the left) and allows the web app to grade the container directly. uAssign design means that students don't have to learn an new interface (something they value according to the survey), but necessitates the use of a grading binary installed in the container for our current implementation.

In addition, TuxLab gives students access to its containers differently than uAssign. TuxLab's Docker containers run in Docker Swarm. Students connect to containers via SSH (with an in-browser SSH client) to an SSH proxy on the TuxLab server, which then connects to the container. This is required due to the way Swarm operates, abstracting containers to "services", where the only interaction with a container's I/O is through Docker-collected logs or though the Internet to a server running in that service, at an IP address in the Swarm network that is essentially randomized. A consequence of this design is that in the current implementation, TuxLab containers continue to have Internet access even after setup, which opens up assignment instances to unwanted capabilities such as package downloads (usurping any instructor-defined setup), communication with outside help, or even malicious servers. Also, since uAssign controls container lifetimes (instead of Swarm), students can navigate away from a problem and return to their container with all of their work still saved.

5 CONCLUSION AND FUTURE WORK

uAssign's implementation demonstrates that authentic assessments for terminal skills can be built in a scalable and secure manner. uAssign gives instructors the opportunity to add terminal-based content to their courses and computer-based exams [19, 20], without sacrificing flexibility in implementation. Through surveys, it's clear that there is a benefit for students, especially those who have not been exposed to the command line and that the current workflow is positive and worth continuing.

There is, however, work to still be done a few dimensions. First, we'd like to extend the range and depth of assignments that have been developed for uAssign. While our students reported significant growth in terminal skills, most students still didn't feel fully comfortable with the terminal. We believe that additional and more varied practice will help them develop that comfort.

Second, we plan to proliferate these terminal assignments to other programming courses. First we will deploy them in other introductory programming courses at our university that serve other populations. Beyond that, we feel there is an opportunity to develop a service that could be used at other universities.

Third, uAssign could be used to study how students learn terminal skills and identify misconceptions. Because uAssign proxies all input and output from its containers to students, we could log the student actions and perform educational pattern mining and qualitative analysis of these traces.

REFERENCES

[1] [n. d.]. Docker. https://www.docker.com/. Accessed: 2018-04-02.

[2] [n. d.]. hterm. https://chromium.googlesource.com/apps/libapps/+/master/hterm. Accessed: 2018-04-01.

[3] [n. d.]. Limit a container's resources | Docker Documentation. https://docs.docker.com/config/containers/resource_constraints/. Accessed: 2018-04-02.

[4] [n. d.]. TuxLab. http://tuxlab.org. Accessed: 2018-03-18.

[5] [n. d.]. uAssign Github repository. https://github.com/jakebailey/ua.

[6] Jacob Bailey. 2018. *uAssign: Scalable and flexible interactive activities for teaching the UNIX terminal.* Master's thesis. University of Illinois at Urbana-Champaign, http://hdl.handle.net/2142/101068.

[7] K. C. Bandi, A. K. Nori, V. Choppella, and S. Kode. 2011. A Virtual Laboratory for Teaching Linux on the Web. In *2011 IEEE International Conference on Technology for Education*. 212–215. https://doi.org/10.1109/T4E.2011.41

[8] Renzo Davoli. 2004. Teaching Operating Systems Administration with User Mode Linux. In *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITiCSE '04)*. ACM, New York, NY, USA, 112–116. https://doi.org/10.1145/1007996.1008027

[9] Jupyter development team and Ramalingam Saravanan. [n. d.]. terminado. https://github.com/jupyter/terminado. Accessed: 2018-04-01.

[10] Andrew Head, Elena Glassman, Gustavo Soares, Ryo Suzuki, Lucas Figueredo, Loris D'Antoni, and Björn Hartmann. 2017. Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale (L@S '17)*. ACM, New York, NY, USA, 89–98. https://doi.org/10.1145/3051457.3051467

[11] Tyson Kendon and Ben Stephenson. 2016. Unix Literacy for First-Year Computer Science Students. In *Proceedings of the 21st Western Canadian Conference on Computing Education (WCCCE '16)*. ACM, New York, NY, USA, Article 14, 4 pages. https://doi.org/10.1145/2910925.2910930

[12] Brian W. Kernighan and Michael E. Lesk. 1979. LEARN — Computer-Aided Instruction on UNIX. In *UNIX Programmer's Manual*. Vol. 2. https://s3.amazonaws.com/plan9-bell-labs/7thEdMan/v7vol2a.pdf

[13] Joseph Bahman Moghadam, Rohan Roy Choudhury, HeZheng Yin, and Armando Fox. 2015. AutoStyle: Toward Coding Style Feedback at Scale. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*. ACM, 261–266. https://doi.org/10.1145/2724660.2728672

[14] Matthieu Moy. 2011. Efficient and Playful Tools to Teach Unix to New Students. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education (ITiCSE '11)*. ACM, New York, NY, USA, 93–97. https://doi.org/10.1145/1999747.1999776 Source code/wiki at http://matthieu-moy.fr/spip/?Unix-training-a-set-of-tools-to.

[15] Fiona Fui-Hoon Nah. 2004. A study on tolerable waiting time: how long are Web users willing to wait? *Behaviour & Information Technology* 23, 3 (2004), 153–163. https://doi.org/10.1080/01449290410001669914 arXiv:https://doi.org/10.1080/01449290410001669914

[16] Lawrence Osborne. 1992. Teaching C with UNIX for College Credit to Professional Programmers. *SIGCSE Bull.* 24, 4 (Dec. 1992), 43–48. https://doi.org/10.1145/141837.141852

[17] Mark Seger. [n. d.]. collectl. http://collectl.sourceforge.net/. Accessed: 2018-04-02.

[18] Matthew West, Geoffrey L. Herman, and Craig Zilles. 2015. PrairieLearn: Mastery-based online problem solving with adaptive scoring and recommendations driven by machine learning. In *2015 ASEE Annual Conference & Exposition*. ASEE Conferences, Seattle, Washington. https://peer.asee.org/24575.

[19] Craig Zilles, Robert Timothy Deloatch, Jacob Bailey, Bhuwan B. Khattar, Wade Fagen, Cinda Heeren, David Mussulman, and Matthew West. 2015. Computerized Testing: A Vision and Initial Experiences. In *2015 ASEE Annual Conference & Exposition*. ASEE Conferences, Seattle, Washington. https://peer.asee.org/23726.

[20] C. Zilles, M. West, D. Mussulman, and T. Bretl. 2018. Making Testing Less Trying: Lessons Learned from Operating a Computer-Based Testing Facility. In *Frontiers in Education*. http://zilles.cs.illinois.edu/papers/zilles_cbtf_fie_2018.pdf