

On-Line Tutorials to Move Students from PC/Windows to Unix

Jennifer Burg and Mark Sherriff
Department of Computer Science
Wake Forest University
Winston-Salem, NC 27109
burg@cs.wfu.edu

Abstract: The majority of today's college students enter their first computer science course with a working knowledge of computers, primarily PC/Windows based. Their experience with PC systems as their "native language" is then reinforced in the many universities that require laptop PCs or distribute them to incoming freshmen. Teaching students the Unix environment thus has become more difficult as students cling to the operating system that they have grown comfortable with. This paper describes a tutorial program intended to move students from PC/Windows to Unix. The self-paced tutorial lessons are written in Macromedia Director and can be accessed from the students' personal computers on the Web. The lessons begin with basic Unix commands executed through a simulated terminal window, provide remote access to an actual Unix terminal window for "real" feedback, direct students to a Unix lab at our university, proceed through more advanced Unix utilities, and offer feedback through graded quizzes.¹

The Problem

The growing number of laptop-equipped students in today's universities is a mixed blessing for the computer science discipline. Students entering our courses believe themselves to be quite computer-savvy, but most have been brought up on PCs/Windows and have a natural reluctance to move out of the environment that has grown so familiar to them. Their portable computers also spoil them on working in their dorms and apartments, and Unix labs that used to bustle with student activity are now mostly quiet.

Most academic computer scientists, however, continue to see the value in teaching students the Unix operating system, and for many good reasons – its robustness; its groundbreaking position as a system for multitasking, networked, and heterogeneous computing environments; and the source-code transparency that lends itself to learning operating system concepts and implementation.

A second problem in introducing students to the Unix environment is that user-level skills don't seem worthy of classroom time, yet we want our students to come out of our courses with a certain level of proficiency in using the operating system. We have found that our students' system skill levels have been dropping in recent years – due to their reluctance to abandon their PC's and spend time in the Unix lab. We need a way to ensure minimum competency in Unix lab skills without sacrificing classroom instruction time to recite simple Unix commands that students could learn on their own.

Thus, the purpose of our tutorial system is to wean PC-devoted students to the Unix environment by letting them get their first glimpse of it through their own personal computers, and to do this in a self-paced tutorial system with student accountability via quizzes and a database that records student activity.

The presentation is user-level – that is, students are taught basic Unix commands, use of text editors, program compilation, and configuration of the user environment. The tutorial system can be used by students receiving their first introduction to the Unix environment, and it is being extended to lessons at a more advanced level, including scripts and shell programming.

¹ This work was supported by a Charles E. Culpeper Grant, the Summer Program for Technology Adoption, at Wake Forest University, Summer 2001.

Related Work

Many Unix tutorials can be found on the Web. However, most of them are text-based rather than interactive. Examples include the tutorials at Idaho State University (<http://www.isu.edu/departments/comcom/unix/workshop/unixindex.html>) and the one at the University of Surrey (<http://www.ee.surrey.ac.uk/Teaching/Unix/unix1.html>). Fairly extensive tutorials are the CERN Unix User Guide at http://consult.cern.ch/writeup/unixguide/unix_2.html and Indiana University's Introduction to the Unix Operating System at <http://www.uwsg.iu.edu/usail/external/riceinfo/UNIX1/unix1.html>.

Among the relatively few interactive tutorials is one by Oregon State University's Computational Physics Department (<http://www.nacse.org/unix-tutorial/>). Students with user accounts can view the tutorial explanations in one window and try them out in a neighboring window. On-line quizzes are also provided. However, this tutorial is available on-line only to OSU students.

A commercial, DOS-based version of an interactive Unix tutorial called L-Nix is available from CyberMatrix [2]. A demo copy is downloaded from <http://www.cyber-matrix.com/lnix.htm>.

We have not encountered any interactive Unix tutorials that emphasize the transition from PC/Windows to Unix.

The Tutorial

Our tutorial system is accessible from the Web at <http://www.cs.wfu.edu/~burg/UnixTutorials/Tutorial.htm>.

We have so far implemented and tested seven tutorial lessons:

- Lesson 1. Unix and the Unix Lab*
- Lesson 2. Getting Started*
- Lesson 3. Basic Unix Commands*
- Lesson 4. Creating Text Files*
- Lesson 5. Using Unix from Your PC*
- Lesson 6. Compiling and Running C++ Programs*
- Lesson 7. File Structure and Access*

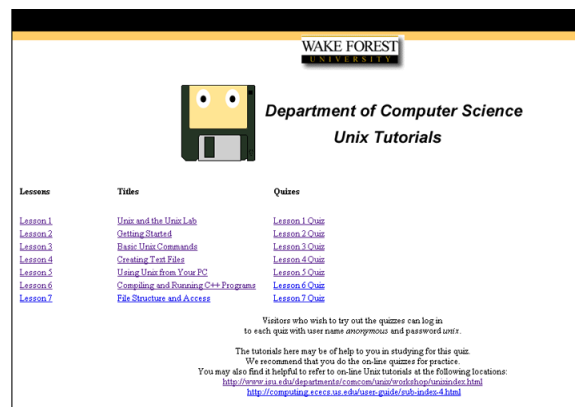


Figure 1. Tutorial Launcher Page

To clarify the difference between working directly on Unix-based computers and remotely from PCs, the first tutorial gives the students a visual tour of the Unix lab at their university. (This is the only lesson specific to our university and its labs.) Students learn where the lab is located, how to get access, and how to get an account on our system. This lesson also touches on what an operating system is and how Unix relates to Windows in the computing world, along with a quick explanation of the hardware and software that we have for them to use.

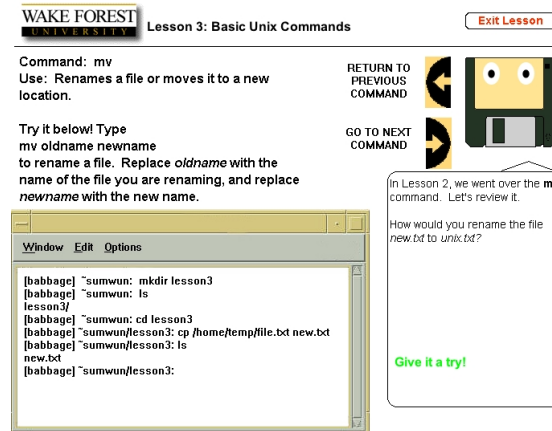


Figure 2. Basic Unix Commands

The second and third lessons, covering basic Unix commands, are presented by means of a simulated Unix terminal window. The students are given practice with simple Unix commands in a controlled environment where errors can be checked and feedback can direct their learning. A character called “Floppy” (reminiscent of “Clippy©” from Microsoft Office) guides the students through the lessons.

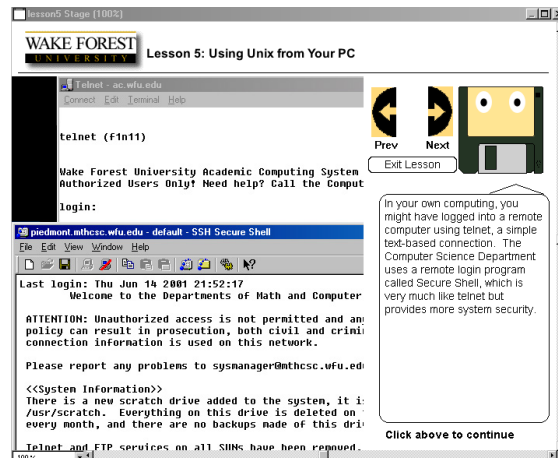


Figure 3. Using Unix from Your PC

Lesson 4 discusses the use of text editors in a Unix terminal window. We chose to use the vi text editor. While this may not be the easiest to learn, it has the advantage of not requiring X-Windows. The students are guided step-by-step throughout the process of using the editor.

In Lesson 5, we begin to ween the students from their PCs by showing them how to get remote access to an actual Unix terminal window. They are given directions on downloading and installing a remote login program, which then opens a terminal window alongside the tutorial. At the end of Lesson 5, students are given a series of exercises to try in the Unix terminal window, some of which will have them investigate new commands, command options, and error conditions more closely. For example, they are asked to remove a directory that still contains files, or they are asked to find an option for the ls command that lists even “hidden files.”

Lesson 6 introduces the use of compilers. C/C++ is chosen as their introductory language since it is the language most closely related to the Unix operating system.

Lesson 7 gives more detail on Unix file structure and how to set permissions on file access.

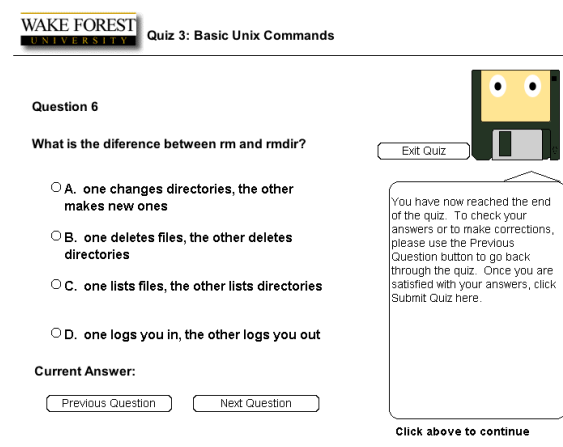


Figure 4. Quiz 3

Each lesson ends in a 10-item quiz. At the end of the quiz, the students can, if they choose, step through the items they got wrong and see the correct answers. The quiz scores are written back to a database for the professor's reference. For our purposes, quiz results were used for assessment of the tutorial system rather than for student grades, as described below.

Implementation

We chose Macromedia Director as the implementation language for the tutorial system for a number of reasons. Director's frame-based programming format lends itself well to computer-based training courses, where there is a progression through an assignment or a quiz. Other important considerations were the ease of building a graphical interface in which the Unix terminal window can be simulated, and the ability to make the tutorials Web-accessible.

An advantage of our implementation method was that it was easy to add new lessons and quizzes once the first had been created since, in Director, one movie can act as a template for others. We created one movie for quizzes, and from that it was only necessary to make a copy, and then update the Lingo code to reflect the new questions and answers. This template-style of implementation was more difficult in the case of the lessons because the types of commands being demonstrated varied greatly. However, we were able to reuse code and frames from one lesson to another.

Like all Web-targeted languages, Director/Shockwave has security constraints on file I/O that we had to deal with. There were two situations in which we wanted to write to files. In Lesson 5, we wanted to offer a dynamic link through which students could install a remote login shell program and an X-windows emulator. These programs would have to be written to the user's hard drive. However, Director/Shockwave restricts file output to one folder pre-set by the system, and writing to this folder would have placed the executable files for the remote login and X-windows program in an unlikely place. (So we abandoned the idea of the dynamic link.) The second area requiring file I/O was in the recording of on-line quizzes. This was less of a problem, since in this case the students' scores are written not to the students' own computers but to the Web server from which the tutorials are run. This is done with a CGI/Perl script that is easily interfaced to the Director program.

Assessment of the Tutorial System

Test Group and Measurements

We have evaluated the effectiveness of our tutorial system on a class of 16 CS II students covering the first five lessons. Our first round of assessment has focused on (1) ensuring the system's robustness and ease of

use; (2) evaluating how useful the lessons and self-quizzes were to students as they prepared for a skills test; and (3) evaluating student study preferences in learning Unix skills.

A preliminary questionnaire was administered to the CS II students during their first class period, asking them about their prior knowledge of Unix. Students were also asked to study a set of Unix commands and told that they would be given a basic skills test the following week. They were then briefly shown how to use the tutorial system so that they could work independently in the self-paced environment. They were told that the lessons and self-quizzes were “voluntary,” and that they would be graded on a skills test administered in class, but would not be graded on their use of the tutorial.

The skills test contained 20 questions on basic Unix commands. A questionnaire was administered after the skills test, asking students for their subjective responses regarding the effectiveness of the tutorial.

System Robustness and Ease of Use

Students had little difficulty using the tutorial system. We spent only about five minutes showing them the interface and gave them the Web address, and they were self-sufficient from that point.

The main bugs in the system uncovered in our “beta testing” related to the responses of the simulated Unix terminal window. It was difficult to anticipate the mistakes that students might make and the sequence of keystrokes they might go through as they tried to execute commands. Thus, a considerable amount of error-checking and feedback were required in the lessons. However, the problems we uncovered in our first test group were easily fixed.

The tutorial system will be used with a second (and larger) test group in the spring semester of 2002.

Pedagogical Effectiveness

The main advantages of the tutorial system are that it doesn't require that class time be devoted to hands-on skills at the expense of concepts and problem-solving; it introduces students to basic Unix skills in a structured way with a logical progression of material and immediate feedback; and it gives students the knowledge and tools to access the Unix network in our department from the laptops, which (experience shows) is the way most students work.

In the past, we have taken one of two routes in teaching CS II: either spend class/lab time going over Unix skills with students, or give them lists of Unix commands and tell them that they are responsible for learning them independently. In the former case, class time is wasted on something the students could and should pick up on their own, and generally the students are bored. In the latter case, many students never take the time to learn the skills without the guidance of a structured assignment.

The tutorials help the students by focusing them on a specific learning activity that may be required of them. They know that the program is written such that it monitors their progress, they receive feedback on the correctness of their responses, and thus they are more likely to take time with the assignment.

The tutorials also show students how to access our Unix network remotely, giving them immediate links from which to download the extra tools they will need (a remote login program and an X-windows emulator). This helps the students to set up the environment that they will probably work in most frequently – accessing the Unix network remotely from their laptops in their dorm rooms, the library, or wherever they happen to be on campus.

The students in our test group performed well on the Unix skills test after using the tutorials. Of the 16 students in the test group, 8 reported knowing “little or nothing” about Unix before doing the tutorials, 6 said that they were “somewhat familiar” with Unix, and 2 were “very familiar.” All but one of the students did all of the tutorials that were available to them (even though they were told that the tutorials were “voluntary”). The average score on the quizzes was 89%.

Interestingly, when asked where they would prefer to learn Unix, a small majority of the students (9 out of 16) replied that they would prefer to work directly in the Unix lab. However, our experience with past CS II classes indicates that even with the best of intentions, students no longer spend a great deal of time in the departmental labs, having gotten accustomed to the comfort of working from their apartments and dorm rooms with their laptop computers. It is worth noting that some mandatory lab assignments would probably be

appreciated by the students. Our CS II students do spend at least one lab session per week in a closed Unix lab (where they work on programs, not directly on Unix skills).

Subjectively, we view the biggest disadvantage of the tutorial as the possibility that students will be confused about the environment they are working in. There is already potential for confusion for beginning computer science students even when they are working directly in the Unix lab, and we are adding more levels of indirection or abstraction to their environment with the tutorials. In the Unix lab, students (at least those curious enough to think about it) eventually learn that their home directories are not on the computer where they happen to be seated, but instead are mounted on that computer from a file server that holds all student files. To add to this indirection, in the first lessons of the tutorial students are not actually on the departmental network at all, but in a “pretend” Unix terminal window. (But the tutorial itself is delivered to them from a remote location on the Web via the campus network.) Then they are introduced to a remote login program and an X-window server, and they must understand that through these they are on the Unix network, running programs from a distance, and displaying them on their laptop.

While no students in our test group complained about being confused, it seems clear that they need an explanation of their working environment. The difficulty is in knowing how many of these “black boxes” ought to be explained in the tutorial without diverting students from their primary focus on basic Unix skills. We believe that concepts relating to remote access ought to be explained by the instructor in class. This is in keeping with our goal of covering hands-on skills through the tutorial and leaving broader concepts for in-class explanation and discussion.

Future Work

Refining and debugging will continue as we incorporate the feedback from this beta test. Future work includes putting more detail into existing lessons, particularly the lesson on text editors. Other lessons to follow will cover topics such as script and shell programming, editing configuration files, and using graphical desktop interfaces.

Upperclass computer science majors (not involved formally in the beta testing) have also tried out the tutorial system and have expressed an interest in more advanced material. The advanced lessons will cover features of systems programming, and installing Linux for dual-boot on a PC.

References

- Abrahams, Paul W., and Larson, Bruce R. *Unix for the Impatient*. 2nd ed. New York: Addison-Wesley, 1996.
- Brown, D. G., Burg, J., and Dominick, J. L. (1998). A Strategic Plan for Ubiquitous Laptop Computing. *Communications of the ACM*, 41 (1), 26-35.
- Robbins, Arnold and Gilly, Daniel. *Unix in a Nutshell: A Desktop Quick Reference for SVR4 and Solaris 7*. 3rd ed. Sebastopol, CA: O'Reilly & Associates, 1999.