6-21-1993

Designing a UNIX Novice Interface

Lee K. Seitz

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Part of the Graphics and Human Computer Interfaces Commons

Recommended Citation

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Honors Program

HONORS SENIOR PROJECT APPROVAL FORM

(To be submitted by the student to the Honors Program with a copy of the Honors Project suitable for binding. All signatures must be obtained.)

Name of Candidate: Lee K. Seitz

Department: Computer Science

Degree: Bachelor of Science

Full Title of Project: Designing a UNIX Novice Interface

Approved by:

[Signature]
Project Advisor

Date

[Signature]
Department Chair

Date

[Signature]
Honors Program Director for Honors Council

Date
Designing a UNIX Novice Interface

by

Lee K. Seitz

Computer Science 495
Mr. John Burleson & Dr. Sajjan Shiva
1 June 1993
Designing a UNIX Novice Interface

The object of this project was to make learning the UNIX operating system easy and enjoyable. The principles involved could be applied to teaching most operating systems, but this project concerned itself with the UNIX operating system. The most important question to be answered was how to go about this teaching process. Before answering this question, however, it might be helpful to examine the history of UNIX. In the mid-1960's, AT&T, General Electric, and several universities worked together on developing an interactive, multi-user operating system for the GE 645 called Multics (Bach 1-2). Many computers of the day ran in batch mode. This meant the computers were fed a program, usually on punch cards, and executed that one program. This type of system had several problems. First, it wasted valuable computer resources by not taking full advantage of their speed and power. Second, programmers did not have any interaction with their programs. They simply got a print out of results. Lastly, it made sharing information difficult (Sobell 4).

Interactive, multi-user systems were designed to overcome these problems. First, by using a time-sharing system, fewer computer resources are wasted. Second, an interactive system allows programmers to manipulate or even abort their programs while they are running. Lastly, by allowing multiple users at the same time, data sharing becomes much easier.

Despite the benefits that could be obtained, AT&T broke away from the Multics project in the late 1960's. Ken Thompson, of AT&T
Bell Laboratories in Murray Hill, New Jersey, had been working on the project and had also been writing a game called Space Travel. When AT&T pulled out, he acquired a PDP-7 computer and developed what he called the UNIX operating system in order to finish his game. The name "UNIX" was selected as a pun on Multics (Bach 2). This initial version of UNIX was written in assembly language (Bach 2), and contained the best features of Multics and other multi-user operating systems that had already been developed (Sobell 4-5). Later, Thompson got a PDP-11 computer and rewrote UNIX in the C programming language (Bach 2). Today, the UNIX source code is still written in C.

In 1975, Bell Labs offered the UNIX system to educational institutions at a low price. Students learned to use it well and as they made their way into the professional world, so did UNIX. In 1978, the UNIX operating system was ported to many different platforms, thanks to the fact it was written in C (Sobell 5).

Today, the UNIX system is in wide use. The two major versions are AT&T's System V and the Berkeley Software Distribution (BSD), developed by the Computer Science Department of the University of California at Berkeley (Sobell 5-6). Versions exist for many platforms, including IBM compatible personal computers. Furthermore, many of these systems are linked through networks, such as Internet, allowing users to share information around the world. An awesome ability for a system designed in order to play a game.

Now that the history of UNIX is understood, it would also be helpful to expand on the features of UNIX. As mentioned before, UNIX is a multi-user system. This means that a computer using the UNIX operating system can have many users using the system simultaneously.
In reality, however, they are not using the computer at the same time. UNIX gives each user pieces of time during which part of the user's current task is performed. Computers can divide these pieces of time into such small amounts and do them so quickly that it seems to each user as if he has the computer's full attention.

Another benefit of having multiple users is in sharing information. One file can be created so that many users have access to it. This prevents each person from having to have his own copy of the file. It also keeps updating simple because only one file has to be updated instead of many. Since most major software products today are undertaken by teams of programmers, this eliminates much of the communication overhead associated with a project in which each team member has his own, single user computer.

A third benefit of the UNIX operating system's multi-user capability is the ability for users to communicate directly. There are many forms of this. The most popular is probably electronic mail (e-mail). E-mail lets users send messages to one another, regardless of whether the recipient is currently using the computer. Messages are stored in a user's "mailbox" until they are deleted by the receiving user. If a system is connected to a network of some type, users might be able to send e-mail to anyone in the world. Other forms of communication are the talk and write commands. These allow users to type messages directly to each other, if both are currently using the system.

The final major benefit of the UNIX operating system's multi-user capability is multitasking. This allows users to carry out more than one function at a time. For example, a programmer could start a program running as what is called a background task, and then answer his mail
while the program runs. As long as the program does not need the programmer to intercede, it will run normally. This makes UNIX very powerful for those that know how to use it.

That returns to the basic question of this project: How to teach UNIX to a novice user? There are several issues to contemplate. First, what exactly is the end-product supposed to teach? As with many projects, this one began on a grand scale. Everything about UNIX was going to be incorporated into it. Naturally, the focus had to be narrowed.

After the above discussion, one rather obvious feature that needed to be included was communication with other users. Also, UNIX is an operating system and an operating system is useless without files. Therefore, file manipulation must be included. Directory manipulation is an equally important aspect of an operating system.

These three topics are probably the core of UNIX, but how much detail should there be in each section? Considering the large amounts of information that each section contains, too many details would probably confuse the user. The goal is to make UNIX easy to learn, so the amount of detail must be enough to give the user an understanding of the concept, but not so much as to bog him down.

The second issue to consider is what approach to use. In this case, it must be easy for a novice to use, yet informative. After consideration, it was decided a menu structure was the best way to do this. Many people find it hard to sit through very structured sessions at a computer when they feel they could be moving ahead. Conversely, some people are intimidated by anything concerning computers and move ahead slowly and timidly. Therefore, it is best to let the user go through a
minimum of highly structured lessons and instead learn what he wants to know, when he wants to know it. A menu structure is conducive to this type of arrangement.

Another fundamental idea behind the project was to teach the user enough so that he could eventually stop using the program and just use the UNIX command line. In order to achieve this, the UNIX Novice Interface (UNI) did not just have to be easy to use, but also it had to be eventually frustrating enough that the user would graduate to using only the UNIX command line. A menu structure also seemed like a way to do this. By forcing the user to use the menus, it would be slower than simply using the command line.

The initial conception was that the main menu would have three sections: Communication, File Manipulation, and Directory Manipulation. Each of these sections would contain commands related to that section. In some cases, commands would appear in more than one section. For example, the move (mv) command could appear in both Directory Manipulation and File Manipulation.

Once an approach was settled on, the method of implementation had to be decided. An obvious choice was the C programming language because it is the language UNIX, itself, is written in, and most UNIX systems have a C compiler. Another choice was a script for one of the many shells available for UNIX. In the end, it was decided to use a Bourne shell script. A shell script was decided upon because it would allow easy access to any UNIX command. The Bourne shell was decided upon because it is one of the most common shells in use on UNIX. This would allow many users to use the program resulting from this project.
Now that all of the fundamental questions about the project had been answered, implementation could begin and further details could be decided upon. In addition to Communication, File Manipulation, and Directory Manipulation, another category was needed. This section would contain only text files explaining various aspects of UNIX. It was decided to call this sections simply "Learn About UNIX." Furthermore, a fifth section was added for Miscellaneous commands that did not fit into any of the three major categories.

As implementation began on UNI, further research was done. It had already been decided that the simplest way to implement a menu structure in a Bourne shell script was to use simply the echo command to print the menu and the case command to execute the appropriate function. As this plan was implemented, it was confirmed that this was the accepted way of implementing a menu (Kochan and Wood 225-30).

To make the menu easier to use and implement, numbers were assigned to each command option. In order for the user to use a command, he simply typed the number by that choice and pressed the Enter key. The program would then ask for further information necessary to implement the command, if there was any. Once all information from the user was obtained, the program would show the user what he would type to execute the command from the UNIX command line. This was to help wean the user off UNI and get him to use regular UNIX.

The number zero was reserved in every menu for the option of returning to the previous menu. In the case of the top menu, this option would exit the program. Another option common to all command menus was reading the manual page built in to UNIX for a command.
To access this option, the user would type the number of the command, a question mark, and then Enter.

Later, it was decided to facilitate the transition from UNI to UNIX by allowing the user easy access to the command line. This was done by adding the exclamation point option to all menus. The exclamation point character was selected because it is typically used to access command line functions in UNIX programs. Using this option, the user could use the command line and then return to precisely where he left in UNI instead of exiting and then restarting the program. This would be most helpful during the intermediate stage when the user is not fully confident in his UNIX knowledge and still needs help occasionally.

As implementation continued, the program evolved. Two more main menu options were added, Select Editor and Edit User Files. The Select Editor option simply let the user select which editor (vi, emacs, or gemacs) he wished to use when editing files. The Edit User Files option, itself, evolved. Originally, it was planned to let the user edit his files that would be executed every time he logged in and out. These plans were put aside for two reasons. First, this would generally require more knowledge than the user would get using UNI. Second, there are so many different types of shells for UNIX that no set standard files could be assumed to exist. In the end, only the files read by the finger command were included in this section.

Another feature of UNI had to be put aside due to time constraints. Originally, a plain English help file was to be written for each command option in each menu. As implementation progressed, the number of files that would have to be written greatly increased. Instead, this
information, for the most frequently used commands, was incorporated into the files contained under Learn About UNIX.

As implementation continued, testing began. The programmer, of course, tested his code as he wrote it to ensure it worked properly. He then placed a copy of the working code where other users could access it. Several users were asked to test UNI and make suggestions. Some of these suggestions were used while others were not.

The overall effect of this method was that the programmer was able to improve UNI by getting input from the very type of people that UNI was designed for. It is hard, once a person learns something, to put themselves back in the place of being a novice. This was, perhaps, the most difficult part of the entire project; trying to anticipate the needs of a novice UNIX user.

The process of designing a UNIX novice interface was a long one. Along the way, many questions had to be asked and answered. Some of these answers were facilitated by a knowledge of the history of UNIX. Many just required a thorough knowledge of the UNIX system, itself. It is hoped that UNI will be a useful tool for many novices seeking to become computer literate, or at least UNIX literate.
Works Cited


APPENDIX A

UNI Source Code
#!/bin/sh
# You may freely copy this shell script provided this copyright notice
# remains intact and that this software is not sold for profit.

# Filename: uni
# Author: Lee Seitz
# Environment: Bourne Shell
# Purpose: To provide novice users of UNIX with a quick and easy
# way of getting started.

# SIMPLE PAUSE ROUTINE
pause () {
    echo
    echo -n "Press <RETURN> to continue..."
    read dummy
}

##### LEARN ABOUT UNIX #######

# PRINT THE LEARN MENU
learnmenu() {
    clear
    echo
    echo
    echo "Learn About UNIX Menu"
    echo " 1. Introduction"
    echo " 2. Conventions of UNIX"
    echo " 3. UNIX vs. DOS"
    echo " 4. Directories and Files"
    echo " 5. Metacharacters"
    echo " 6. Pipes and Redirection"
    echo " 7. Processes"
    echo " 0. Return to main menu"
    echo " !. Shell"
    echo -n "Your choice?"
}

### LEARN ABOUT UNIX ###

learn() {
    topic=20
    while test "$topic" != "0"
do
learnmenu
read topic
clear
case $topic in
  0) echo ;;
  1) more $UNI/intro.uni ;;
  2) more $UNI/conventions.uni ;;
  3) more $UNI/unix_v_dos.uni ;;
  4) more $UNI/dirs_and_files.uni ;;
  5) more $UNI/metachar.uni ;;
  6) more $UNI/pipes_and_redir.uni ;;
  7) more $UNI/process.uni ;;
  !) echo "Type CTRL-D or exit to return to UNI."
     $SHELL
     echo "Returning to UNI..."
     topic=20
      ;;
  *) echo
      echo "Error: Please try again."
      ;;
esac
if test "$topic" != "0"
  then
    pause
  fi
done
}

### FILE MANIPULATION ###

# PRINT THE FILE MENU
filename() {
  clear
echo
echo "File Manipulation Menu"
echo
echo -n "Current directory:"
pwd
echo
echo " 1. List files   (ls)"
echo " 2. Change directory   (cd)"
echo " 3. View file   (more)"
echo " 4. Edit file   ($EDITOR)"
}
echo " 5. Execute file"
echo " 6. Copy files (cp)"
echo " 7. Rename (move) file (mv)"
echo " 8. Delete (remove) file (rm)"
echo " 9. Create links (ln)"
echo " 10. Change file permissions (chmod)"

# LIST FILES
list() { 
  clear
  echo
  echo "    List Files"
  echo 
  echo "Usage: ls [-aAcdfgIlqRrstul] filename ..."
  echo 
  echo "Suggested options:"
  echo "    -F (mark directories (/), executables (*), and links (@))"
  echo "    -C (columns)"
  echo 
  echo -n "Current directory:"
  pwd 
  echo 
  echo -n "Directory (<RETURN> for current)? "
  read dir 
  echo -n "Options (Include dash; <RETURN> for none.)? "
  read opts 
  echo 
  echo "You would type: ls $opts $dir | more"
  echo 
  ls $opts $dir | more
  pause 
}

# CHANGE DIRECTORY
chandir() { 
  clear
  echo
echo "   Change Directory"
echo echo "Usage: cd [directory]" echo echo "Common directories:"
  echo "   .. (parent)"
  echo "   <none> (home)"
  echo "   / (root)"
  echo "   . (current)"
  echo echo -n "Current directory:"
pwd echo echo -n "Directory?"
read dir echo echo You would type: cd $dir if test -d ${dir:=$HOME} then cd $dir else echo echo "$dir: No such file or directory" fi pause

# VIEW FILES
view() { clear echo echo "   View Files"
echo more echo echo -n "Filename (Include pathname if not current.)?"
read source echo echo You would type: more $source
  more $source
  pause
}

# EDIT FILES
edit() { clear
echo "Edit Files"
read source
echo You would type: $EDITOR $source
pause
$EDITOR $source
#
# EXECUTE FILES
run() {
  clear
  echo "Execute Files"
  echo -n "Filename (Include pathname if not current.)? "
  read source
  echo You would type: $source
  pause
  $source
}
#
# COPY FILES
copy() {
  clear
  echo "Copy Files"
  cp
  echo "Suggested option:"
  echo " -i (interactive)"
  echo -n "Source file(s) and/or directory? "
  read source
  echo -n "Destination? "
  read dest
  echo -n "Options? "
  read opts
  echo You would type: cp $opts $source $dest
  cp $opts $source $dest
  pause
# RENAME FILES
move() {
    clear
echo
echo " Rename (Move) Files"
echo
echo mv
    echo
echo "Suggested option:"
    echo " -i (interactive)"
    echo
echo -n "Filename (Include pathname if not current.)? "
    read source
echo -n "New name (Include pathname again, if necessary.)? "
    read dest
echo -n "Options? "
    read opts
echo
echo You would type: mv $opts $source $dest
    mv $opts $source $dest
pause
}

# DELETE FILES
remove() {
    clear
echo
echo " Delete (Remove) Files"
echo
echo rm
    echo
echo "Suggested option:"
    echo " -i (interactive)"
    echo
echo -n "Filename (Include pathname if not current.)? "
    read source
echo -n "Options? "
    read opts
echo
echo You would type: rm $opts $source
    rm $opts $source
    pause
}
# CREATE SOFT OR HARD LINKS

link() {
  clear
  echo
  echo "    Create Links"
  echo
  echo
  echo -n "File or directory name with complete pathname? "
  read source
  echo -n "Link name (<ENTER> to keep above name.)? "
  read lnkname
  echo "Pick the type of link:"
  echo "    1. Hard"
  echo "    2. Soft"
  echo "    0. Abort!"
  echo
  echo -n "Which? "
  read lnknum
  case $lnknum in
    1) echo You would type: ln $source $lnkname
      ln $source $lnkname
      ;;
    2) echo You would type: ln -s $source $lnkname
      ln -s $source $lnkname
      ;;
    *)
  esac
  pause
}

# CHANGE FILE PERMISSIONS

chmod() {
  clear
  echo
  echo "    Change Permissions"
  echo
  echo "WARNING: You could accidentally remove your own permission"
  echo "to execute, read, or write one of your files or directories."
  echo "To abort, press <RETURN> at the next two prompts."
  echo
  chmod
  echo
  echo -n "Filename (Include pathname if not current.)? "
  read source
}
echo -n "Options?"
read opts
echo You would type: chmod $opts $source
chmod $opts $source
pause
}

# *** FILE MANIPULATION ***
filemanip() {
    num=20
    while test "$num" != "0"
do
        filemenu
        read num
        case $num in
            0) echo ;;
            1) list ;;
            2) changdir ;;
            3) view ;;
            4) edit ;;
            5) run ;;
            6) copy ;;
            7) move ;;
            8) remove ;;
            9) link ;;
            10) chmodmod ;;
            "1?") clear
                echo You would type: man ls
                man ls ;;
            "2?") clear
                echo You would type: man cd
                man cd ;;
            "3?") clear
                echo You would type: man more
                man more ;;
            "4?") if test $EDITOR = "emacs"
                then
                    echo
                    echo "Sorry, no man page for emacs."
                else
                    clear
                    man $EDITOR
                fi
            "5?") echo "Sorry, no man page for this." ;;
            "6?") clear
        esac
    done
}
echo You would type: man cp
man cp

"7?"
    clear
    echo You would type: man mv
    man mv

"8?"
    clear
    echo You would type: man rm
    man rm

"9?"
    clear
    echo You would type: man ln
    man ln

"10?"
    clear
    echo You would type: man chmod
    man chmod

!
    clear
    echo "Type CTRL-D or exit to return to UNI."
    "$SHELL"
    echo "Returning to UNI..."
    num=20
    *
    echo "Error: Please try again."
    pause
    esac
    case $num in
    *"
    esac
    done
}

##### DIRECTORY MANIPULATION

# PRINT THE FILE MENU
dirmenu() {
  clear
  echo
  echo
  echo "Directory Manipulation Menu"
  echo
  echo -n "Current directory:"
  pwd
}
echo
echo " 1. Change directory   (cd)"
echo " 2. Make directory    (mkdir)"
echo " 3. Remove directory   (rmdir)"
echo " 4. Create links       (ln)"
echo " 5. Change dir. permissions (chmod)"
echo " 6. List files         (ls)"
echo " 7. List current directory (pwd)"

for i in 0 1 2 3 4 5 6 7
  do
    echo " 0. Return to main menu"
    echo " 1. Shell"
    echo " #?. Read man page on #."
    echo " #. Read man page on #."
    echo " Your choice? 

    creatdir() {
      clear
      echo
      echo " Make Directories"
      echo mkdir
      echo
      echo -n "Directory name? 
      read newdir
      echo
      echo You would type: mkdir $newdir
      mkdir $newdir
      pause
    }
  
    removdir() {
      clear
      echo
      echo " Remove Directories"
      echo rmdir
      echo
      echo -n "Directory name (Include pathname if not current.)? 
      read source
      echo
      echo You would type: rmdir $source
rmmdir $source
pause
}

# CURRENT DIRECTORY
currdir() {
clear
echo
echo "You would type: pwd"
pwd
echo
pause
}

# *** DIRECTORY MANIPULATION ***
dirmanip() {
num=20
while test "$num" != "O"
do
dirmenu
read num
case $num in
  0) echo ;;
  1) changdir ;;
  2) creatdir ;;
  3) removdir ;;
  4) link ;;
  5) changmod ;;
  6) list ;;
  7) currdir ;;
  "1?" ) clear
        echo You would type: man cd
        man cd ;;
  "2?" ) clear
        echo You would type: man mkdir
        man mkdir ;;
  "3?" ) clear
        echo You would type: man rmdir
        man rmdir
;;
  "4?" ) clear
        echo You would type: man ln
        man ln
;;
  "5?" ) clear
        echo You would type: man chmod
man chmod
!
$ clear
  echo "Type CTRL-D or exit to return to UNI."
  $SHELL
  echo "Returning to UNI..."
  num=20

*) echo "Error: Please try again."
  pause

esac
  case $num in
  *"?"*) pause ;;
  esac
done
}
echo " 2. What is who doing? (w)"
echo " 3. Finger a user. (finger)"
echo " 4. Write a user. (write)"
echo " 5. Talk to a user. (talk)"
echo " 6. Receive and send mail. (elm)"
echo " 7. Brief list of users. (users)"
echo " 8. List of remote users. (rusers)"
echo " 9. Who am I? (who am i)"
echo "10. Allow/Disallow Write & Talk. (mesg)"
echo "11. Mail Notification On/Off. (biff)"

# WHO IS ON
whoson() {
    clear
    echo
    echo "You would type: who | more"
    who | more
    echo
    pause
}

# WHAT ARE OTHERS DOING
what() {
    clear
    echo
    echo "What's Who Doing?"
    echo
    echo "Usage: w [-hls] [user]"
    echo
    echo -n "Username (<RETURN> for all)? "
    read u
    echo -n "Options? "
    read o
    echo
    echo "You would type: w $o $u | more"
    w $o $u | more
    echo
# FINGER USERS

def fngr():
    clear
    echo
    echo "   Finger"
    echo
    echo "Usage: finger [-bfhilmqsw] [name]"
    echo
    echo -n "Name(s) (<RETURN> for all current users)? "
    read names
    echo -n "Options? "
    read opts
    echo
    echo "You would type: finger $opts $names \ more"
    finger $opts $names \ more
    echo
    pause

# WRITE USER

def wrt():
    clear
    echo
    echo "   Write"
    echo
    echo "After giving this command, whatever you type will be displayed"
    echo "on the other users terminal each time you hit <RETURN>."
    echo "Hit <CTRL>-D to quit."
    echo
    write
    echo
    echo -n "Username (<RETURN> to abort)? "
    read usname
    echo
    echo "You would type: write $usname"
    write $usname
    echo
    pause

# TALK TO A USER

def tlk():
    clear
echo "Talk"

echo "After giving this command, you will have to wait for the other user to respond. When he does, whatever is typed will appear on both terminals. Your messages will appear on the top half of the screen and the other users will appear on the bottom half." 

echo "Hit <CTRL>-C to quit."

echo

talk

echo -n "Username[@host] (<RETURN> to abort)? "
read usname

echo "You would type: talk $usname"

talk $usname

echo

pause

# MAIL (ELM)
domain() {
    clear
echo
    echo "Elm"
    echo
    echo "You would type: elm"
    pause
    elm
}

# USERS
domain() {
    clear
echo
    echo You would type: users
users
echo
    pause
}

# REMOTE USERS
domain() {
    clear
echo
    echo "Remote Users"
echo

"Usage: rusers [-ahilu] [host...]

WARNING: This could take several minutes."

read yn
case $yn in
  Y|y)
    echo -n "Options? "
    read opts
    echo You would type: rusers $opts
    rusers $opts
    ;;
  esac
  esac

whomi() {
  clear
  echo
  echo You would type: who am i
  who am i
  echo
  pause
}

msg() {
  mesg > /dev/null
  if [ $? -eq 0 ]
    then
      echo
      echo You would type: mesg n
      mesg n
      pause
    else
      echo
      echo You would type: mesg y
      mesg y
      d pause
    fi
}

bff() {


biff > /dev/null
if [ $? -eq 0 ]
then
  echo
  echo You would type: biff n
  biff n
  pause
else
  echo
  echo You would type: biff y
  biff y
  pause
fi

# *** OTHER USERS ***
others() {
  othernum=20
  while test "$othernum" != "0"
  do
    othermenu
    read othernum
    case $othernum in
    0) echo ;;
    1) whoson ;;
    2) what ;;
    3) fngr ;;
    4) wrt ;;
    5) tlk ;;
    6) domail ;;
    7) usrs ;;
    8) rusrs ;;
    9) whomi ;;
    10) msg ;;
    11) bff ;;
    "1?""9?") clear
      echo You would type: man who
      man who
    ;;
    "2?") clear
      echo You would type man w
      man w
    ;;
    "3?") clear
      echo You would type: man finger
      man finger
"4?") clear
echo You would type: man write
man write

"5?") clear
echo You would type: man talk
man talk

"6?") clear
echo You would type: man elm
man elm

"7?") clear
echo You would type: man users
man users

"8?") clear
echo You would type: man rusers
man rusers

"10?") clear
echo You would type: man mesg
man mesg

"11?") clear
echo You would type: man biff
man biff

! clear
echo "Type CTRL-D or exit to return to UNI."
$SHELL
echo "Returning to UNI..."
othernum=20

*) echo
echo "Error: Please try again."
pause

esac
case $othernum in

=*/?") pause ;;
esac
done
}
edmenu() {
    clear
    echo
    echo
    echo "Select Editor"
    echo
    echo "Current editor: "$EDITOR
    echo
    echo "Pick your favorite editor:"
    echo "  1. vi"
    echo "  2. emacs"
    echo "  3. gemacs"
    echo
    echo "  0. Return to main menu"
    echo
    echo "  !. Shell"
    echo
    echo "  #?. Read man page on #."
    echo
    echo -n " Which? "
}

picked() {
    ednum=20
    while test "$ednum" !="0"
do
    edmenu
    read ednum
    case $ednum in
        0) echo ;;
        1) EDITOR=vi;;
        2) EDITOR=emacs ;;
        3) EDITOR=gemacs ;;
        "1?") clear
               echo You would type: man vi
               man vi
               pause
               ;;
        "2?") echo
               echo You would type: man emacs
               man emacs
               pause
               ;;
    esac
    done
}
"3?") clear
  echo You would type: man gemacs
  man gemacs
  pause
  ;
  !) clear
  echo "Type CTRL-D or exit to return to UNI."
  $SHELL
  echo "Returning to UNI..."
  ednum=20
  ;
  *) echo
  echo "Error: Please try again."
  pause ;;
esac
done
}

##### EDIT FINGER FILES #################################################

# PRINT THE EDIT FINGER FILES MENU
edfilesmenu() {
  clear
  echo
  echo
  echo " Edit User Files"
  echo
  echo "  1. Edit .plan"
  echo "  2. Edit .project"
  echo
  echo "  0. Return to main menu."
  echo
  echo "  !. Shell"
  echo
  echo "  #. Give information on #."
  echo
  echo -n " Which? "
}

# EDIT .plan
editplan() {
  clear
  echo
  echo " Edit .plan File"
  echo
echo You would type the following:
  echo "  cd (To ensure you're in your home directory.)"
  echo "  "$EDITOR" .plan"
  echo "  chmod 644 .plan (To ensure others can read it.)"
  echo pause
  cd
  $EDITOR .plan
  chmod 644 .plan
}
# EDIT .project
editproj() {
  clear
echo " Edit .project File"
echo
  echo You would type the following:
  echo "  cd (To ensure you're in your home directory.)"
  echo "  "$EDITOR" .project"
  echo "  chmod 644 .project (To ensure others can read it.)"
  echo pause
cd
  $EDITOR .project
  chmod 644 .project
}

# *** EDIT USER FILES ***
edfiles() {
edchoice=20
  while test "$edchoice" != "0"
do
    edfilesmenu
    edfilesmenu
    read edchoice
case $edchoice in
  0) echo ;;
    1) editplan ;;
    2) editproj ;;
    1?) clear
      more $UNI/plan.uni
      pause
    ;;
    2?) clear
      more $UNI/project.uni
      pause
    ;;
! clear
echo "Type CTRL-D or exit to return to UNI."
$SHELL
echo "Returning to UNI..."
edchoice=20
;;
*) echo "Error: Please try again."
pause
;;
esac
done
}

### MISCELLANEOUS

# PRINT THE MISC. MENU
miscmenu() {
  clear
echo
echo
  echo "  Miscellaneous"
  echo
echo "  1. Read a man page. (man)"
echo "  2. Find text in file(s). (grep)"
echo "  3. View a file (no pause). (cat)"
echo "  4. See current processes. (ps)"
echo "  5. Kill a process. (kill)"
echo
echo "  0. Return to main menu."
  echo
echo "  #?. Read man page on #."
  echo
echo "  !. Shell"
  echo
echo -n " Which? "
}

# MAN PAGE
manpg() {
  clear
echo
  echo "  Read Man Pages"
  echo
  man
echo
echo -n "Topic? "
  echo
  echo
read topic
echo -n "Options?"
read opts
echo
echo You would type: man $opts $topic
man $opts $topic
pause
}

# FIND TEXT IN FILE
findtext() {
clear
echo
echo "  Find Text in File(s)"
echo
grep
echo
echo -n "Text?"
read text
echo -n "File(s)?"
read file
echo -n "Options?"
read opts
echo
echo You would type: grep $opts $text $file | more
grep $opts $text $file | more
pause
}

# VIEW FILES
view2() {
clear
echo
echo "  View Files"
echo
echo "Usage: cat [-] [-benstuv] [filename...]
" echo
echo -n "Filename (Include pathname if not current.)?"
read source
echo
echo You would type: cat $source
cat $source
pause
}

# SHOW PROCESSES
procs() {
    clear
    echo
    echo You would type: ps I more
    ps I more
    echo
    pause
}

# KILL A PROCESS
killproc() {
    clear
    echo
    echo " Kill a Process"
    echo
    echo "Usage: kill [-signal] pid ..."
    echo
    echo -n "Process id? "
    read procid
    echo -n "Options? 
    read opts
    echo
    echo You would type: kill $opts $procid I more
    kill $opts $procid I more
    pause
}

# *** MISCELLANEOUS ***
misc() {
    miscchoice=20
    while test "$miscchoice" != "0"
    do
        miscmenu
        read miscchoice
        case $miscchoice in
            0) echo ;;
            1) manpg ;;
            2) findtext ;;
            3) view2 ;;
            4) procs ;;
            5) killproc ;;
            !) clear
        echo "Type CTRL-D or exit to return to UNI."
        $SHELL
        echo "Returning to UNI..."
        miscchoice=20
    esac
}
"1?") clear
echo You would type: man man
man man
pause

"2?") clear
echo You would type: man grep
man grep
pause

"3?") clear
echo You would type: man cat
man cat
pause

"4?") echo
echo You would type: man ps
man ps
pause

"5?") echo
echo You would type: man kill
man kill
pause

*) echo "Error: Please try again."
pause
esac
done
}

### MAIN MENU

# PRINT THE MAIN MENU
mainmenu() {
    clear
echo
    echo
    echo "UNIX Novice Interface (UNI)"
echo "MAIN MENU"
echo
    echo "1. Learn about UNIX"
echo "2. File Manipulation"
echo "3. Directory Manipulation"
echo " 4. See/Communicate with Other Users"
echo " 5. Edit User Files"
echo " 6. Select Editor (Currently "$EDITOR")"
echo " 7. Miscellaneous"
  
echo
  
  0. Exit
  
  !. Shell
  
echo -n " Your choice? "
}

# *** MAIN PROGRAM ***
UNI=`pwd`
clear
echo
  
  UNIX Novice Interface (UNI)"
  
  Copyright 1992, 1993 by Lee Seitz"
  
  UNIX is a registered trademark of AT&T."
  
  Do you wish to read the introduction (y/n)? "
read yn
case $yn in
  Y|y) clear
      more intro.uni
      echo
      pause
      ;;
  esac
choice=20
while test "$choice" != "0"
do
  trap "2
  mainmenu
  read choice
case $choice in
    0) echo -n " Are you sure (y/n)? "
        read yn
        if test "$yn" = "y" -o "$yn" = "Y"
          then
          echo
        else
          choice=20
        fi
  esac
  done
1) learn ;
2) filemanip ;
3) dirmanip ;
4) others ;
5) edfiles ;
6) picked ;
7) misc ;
!

clear

echo "Type CTRL-D or exit to return to UNI."
$SHELL

echo "Returning to UNI..."
choice=20

*)

echo "Error: Please try again."
pause

esac
done
trap 2
clear
echo "Thank you for using UNI."
UNIX Novice Interface (UNI)

Welcome to UNI! This program is designed to introduce you, a novice UNIX user, to the UNIX operating system. Hopefully you will learn enough here to become an eXpert, and thus use UNIX instead of UNI.

Before going any further, let's start on your first lesson right now. You are viewing this text file with a common UNIX utility called "more." More allows you to view text files one screen at a time. You will notice a line saying "--More--" at the bottom of the screen, indicating that there is more text in this file to view. You have several options, but here are the some of the important ones:

- **RETURN**: Pressing return advances one line.
- **SPACE**: Pressing the spacebar advances one screen.
- **"?"**: Pressing ? gets help on more.
- **"s"**: Pressing s skips ahead one screen.
- **"k"**: Pressing k skips ahead one line.
- **"b"**: Pressing b goes back one screen.
- **"q" or "Q"**: Pressing q quits more.

If you press a number before the s, k, or b commands, it will skip that many lines or screens.

You might also notice that sometimes more will give you a percentage at the end of its prompt. This tells you how much of the document you have viewed so far. Go ahead and play with it a little, but do not hit "Q" or go past the end of this text until you are ready to go to the UNI menu system. If you do accidentally finish the document, you may read it again in the Learn about UNIX section of the main menu.

Now, some more about UNI. I am assuming that you have some computer experience, and thus are familiar with some type of operating system, probably MS-DOS or Windows. This means you understand, to some extent, command line options, files, directories, and so on. If you are not familiar with these terms, you should read all of the files under Learn about UNIX. You should also supplement your knowledge with a book on UNIX. _A Practical Guide to the UNIX System_ by Mark G. Sobell (Redwood City, CA: The Benjamin/Cummings Publishing Co., Inc., 1989.) is recommended, but any UNIX book should do.
CAUTION: One more screen to the end!

This is the end!!

MS-DOS and Windows are trademarks of Microsoft Corporation.
UNI Conventions

UNI is made up of several menus and prompts. Each menu is made up of a set of numbered items. To choose an item, simply type its number and press <RETURN> (also known as <ENTER>). If the item is a UNIX command, the command will appear in parentheses after the description of the command in the menu. If the item requires multiple commands, these will be listed when you select the item.

Some selections on each menu are consistent throughout all menus. First, the zero (0) option will always take you back one level. If you are at the main menu, this option will exit UNI. Only at this, the top level, will UNI confirm that this is what you want to do. Second, on most menus, if you type the number of an item followed by a question mark (?) and then hit <RETURN>, a help file will appear for the item. In the case of UNIX commands, these will be the manual pages built in to UNIX.

Another option available in all menus is the shell option. This option is executed by typing an exclamation point (!) and <RETURN>. This allows you to use the command line without actually quitting UNI. In this manner, if you wish to practice using UNIX, you can do so, but still have an easy way back to UNI if you get stuck. To return to UNI after using this option, type "exit" and press <RETURN> or hold down the CONTROL key and press D (noted as CTRL-D or ^D).

Before a command is carried out, UNI will tell you what it is that you would type at the UNIX prompt if you wanted to execute the command. This is the method by which you should learn how to use UNIX. Pay attention to these messages and try to remember them. As you do, you'll start learning how to use UNIX from the command line, which is much more powerful than UNI. Your ultimate goal should be to not need UNI for your UNIX work.

Another feature of UNI is that all output that might be longer than your screen is piped into the more utility. (For more on pipes and more, see the section on redirection & pipes and the UNI introduction, respectively.) Note that this pipe into more will also appear when UNI tells you what you would type at the command line. If the output is not longer than the screen, the more prompt will not appear despite the pipe. At the end of all output, no matter how long it is, UNI will pause and wait for you to hit <RETURN> before returning to the current menu. As a matter of fact, this is the end of this file, so you should be seeing that message now.
UNIX vs. DOS

If you are familiar with MS-DOS or PC-DOS, this section is for you. Here, the differences between UNIX and DOS will be discussed. UNIX and DOS have many things in common, because much of DOS was derived from UNIX. Even if you don't have any DOS experience, you might find some interesting notes here.

The most important note is that, unlike DOS, UNIX is case sensitive. Virtually all UNIX commands should be typed in lower case. The case sensitivity plays its biggest role in command options. For example, "ls -a" is only slightly different from "ls -A", but using "-c" instead of "-C" with the "ls" command will give completely different results.

The next note to make is that the directory structures of the two operating systems are virtually the same. The cd command is also virtually identical. The crucial difference is that UNIX uses a slash (/) instead of a backslash (\). Furthermore, you must put a space between cd and the directory, even if it begins with a slash. Finally, there is no distinction between disk drives in UNIX (i.e. A:, B:, C:, etc).

Other minor differences are in filenames and metacharacters (wildcards). UNIX filenames are not restricted to eight characters, a period, and three more characters, as in DOS. They may contain 14 characters in System V or 255 characters in BSD. These characters can be almost any character on the keyboard. If a filename begins with a period (.), it will not show up in the regular directory listing (the ls command). To reveal these files, the -a option must be used (i.e. "ls -a").

Because there is not set format for UNIX filenames, the asterisk (*) metacharacter refers to all files, if used by itself. This makes "rm *" a very dangerous command, analogous to "del *.*". Characters can be placed before and/or after the asterisk to limit its scope (e.g. "rm *.txt.1", ). The question mark (?), works just as in DOS, matching any one character.

Here is a summary of common DOS commands and the corresponding UNIX commands. Commands in parentheses indicate abbreviations for the command. Note that the UNIX command might not work exactly like the DOS command. Furthermore, if a DOS command has a set of options, they will be different or non-existent in UNIX. UNIX options are preceded by a dash (-), rather than a slash (/). For more help, read the man page on each command (e.g. "man mv").

<table>
<thead>
<tr>
<th>DOS</th>
<th>UNIX</th>
</tr>
</thead>
<tbody>
<tr>
<td>chdir (cd)</td>
<td>chdir (cd)</td>
</tr>
<tr>
<td>command</td>
<td>sh OR csh OR ... (Depends on system.)</td>
</tr>
<tr>
<td>copy</td>
<td>cp (OR ln)</td>
</tr>
<tr>
<td>date</td>
<td>date</td>
</tr>
<tr>
<td>del (erase)</td>
<td>rm</td>
</tr>
<tr>
<td>dir</td>
<td>ls</td>
</tr>
<tr>
<td>exit</td>
<td>exit (CTRL-D)</td>
</tr>
<tr>
<td>fc</td>
<td>cmp</td>
</tr>
<tr>
<td>find</td>
<td>grep</td>
</tr>
<tr>
<td>help</td>
<td>man</td>
</tr>
<tr>
<td>mkdir (md)</td>
<td>mkdir</td>
</tr>
</tbody>
</table>
move
rename (ren)
rmdir (rd)
sort
time
type
type <fname> | more

mv
mv
rmdir
sort
date
cat
more OR page (Depends on system.)

Note that there is no rename command in UNIX. Instead, the move command is used. If you wished to rename the file a file called This in the current directory to That, you would type "mv This That". If, in addition to renaming it, you wanted to move it from the current directory to a subdirectory called backup, you would type "mv This ./backup/That". To move it without renaming it, you would type "mv This ./backup".

Also note that the UNIX "date" command only gives the current date and time, it does not usually allow you to change them. You may also wish to try the "cal" command, which prints calendars of months or even entire years. (Examples: cal 1993, cal 1 2001.)

Lastly, you may actually wish to simply link a file instead of copying the entire file. See the UNIX Directories, Files and Links document for more details.

MS-DOS is a trademark of Microsoft Corporation.
PC-DOS is a trademark of IBM.
UNIX Directories, Files, and Links

A directory refers to a set of files, links, and other directories (called subdirectories). The UNIX directory structure is often defined as a tree. The root directory (designated by a /) is the topmost directory. All other directories are underneath it, branching out from it.

**cd command:**

To change from one directory to another, use the cd command. The form is "cd directory". Directory may be just a subdirectory name, or a complete path name. For instance, if you were in the root directory and you wanted to change to the pub directory, you would type "cd pub". If, from there, you wanted to change to the etc directory, which is also underneath the root directory, you could type "cd /etc". The slash indicates the root directory. It is also used to separate directory names. For example, "cd /usr/people/jsmith".

There are two special types of directory entries in all directories, the single (.) and double periods (..). The single period refers to the current directory. Type "cd ." doesn't actually change your directory. The double period refers to the parent directory of the current directory. For instance, if you're in /usr/people/jsmith, typing "cd .." would change you to /usr/people. These directories are also useful as shorthand. For instance, if you're in /usr/people/jsmith and you want to be in /usr/people/jdoe, you could type "cd ../jdoe".

**ls command:**

To view the contents of a directory, the ls command is used. This will display all the files, links, and subdirectories within the current directory, except those that begin with a period. These are usually system files or directories that the user does not need to worry about. If you wish to see them, also, use the -a option with ls (i.e. "ls -a"). Also, if you wish to only view a few files, you might be able to use metacharacters to help. (e.g. "ls -1 p*.txt"). See the UNIX Metacharacters section for more on them.

Note that the ls command, by itself, only shows the names of the files and directories. In order to view all the information about the files, use the -l option. This will display one file, link, or directory with the following information: its type and permissions, number of links, owner, size, date modified, and name.

**In command:**

The first column is a column of ten characters. This first lets you know the type of entry, usually file (-), directory (d), or link (l). A file is just a sequence of bytes. Files usually contain text, executable or compressed data. Directories were discussed above. A link is a name that refers to a file or directory. This allows users to reference a file from a directory without actually copying the file, thus using less storage space.

To create a link, use the ln command. The format is "ln existing-file new-link". The existing-file is the complete path of the original file and new-link is the (new) name for the file in the current directory. These are called hard links and can only be done on files. Berkeley UNIX allows you to link entire directories using the -s option (i.e. "ln -s existing-file new-link"). This is called a symbolic or soft link.
The next column from the long `ls` output tells the number of links to the entry. For a directory, this is usually once for itself (.), once for its parent (..), and then once for each of its subdirectories. For example, a directory with three subdirectories usually has a link count of five.

Permissions & chmod command:

The next nine characters from the long form of `ls` reveal the entry's permissions. These nine characters can be divided into three groups of three. Each group contains the read (r), write (w), and execute (x) permissions. The three groups are the permissions for the user (u), his group (g), and all other users (o).

For files, the read permission lets the user view the contents of the file, the write permission lets the user rewrite or overwrite the file, and the execute permission lets the user run the file. Note that you can make a file executable even if it UNIX cannot actually execute it.

For directories, the read permission lets the user view the contents of the file, even if they can't enter it; the write permission lets the user create, delete, and remove files, if they also have write permission for the file; and the execute permission lets the user enter the directory and use whatever permissions they have on the files, although they can't get a listing. Usually, if users are to use the directory without changing anything, they are given read and execute permission.

In order to set the permissions for a file or directory, the `chmod` command is used. It has two forms. In the first is "chmod [ugoa][+-][rwx] name". The first letter specifies whether to give or remove the permission for the owning user, his group, other users, or all three sets. The + gives the permission and the - removes it. Finally, the r, w, or x specifies which permission. The name is the file or directory having its permissions changed. The second format is "chmod ### name", where # is a digit 0-7. In this case, think of each group as an octal digit, where the r = 4, w = 2, and x = 1. You simply add up the numbers for each group to give the right permissions. For example, to give yourself all permissions and everyone else none for a file called temp, you would type "chmod 700 temp". If you then decided to let anyone read and execute the file, you would type "chmod 755 temp".
UNIX Metacharacters

Metacharacters are single characters used to stand for many characters or multiple possibilities in a single character. The most often used metacharacters are the asterisk (*) and question mark (?). The asterisk matches zero or more characters in a filename. If used by itself in a command, it refers to all files (e.g. [rm *]). (The brackets in all these examples indicate what is to be typed and are not part of the command.) Characters may be inserted before or after it to limit what files it refers to. For example, [ls *.txt] lists all files ending in ".txt"; [mv letter* ../letters] moves all files starting with "letter" to a directory called letters; and [rm a*.test] removes all files beginning with "a" and ending with ".test".

The question mark matches one character in a filename. For example, [rm test.?] would remove files such as "test.1", "test.2", and "test.a", but not "test.abc".

In order to refer to a file with a metacharacter in it, the backslash (\) is used. For example, if you had the files "abc", "abba", and "a*" and you wished only to delete the file "a*", you would use [rm a\*]. If you used [rm a*], all three files would be deleted.

Another way of designating literal strings is the single quote ('). In the above example, [rm 'a*'], could have also been used. Double quotes designate a literal string, after evaluating $, `, and \. For example, the previous command could have also been [rm "a*"] or even [rm "\a*"].

Lastly, the backquote (`) metacharacter executes the command in backquotes and replaces the command with its output. For example, [echo "ls a*"] would print "ls a*", but [echo `ls a*`] prints all the files that begin with the letter a.
Pipes and Redirection

One feature of UNIX is the ability to redirect output and input for a command, or pipe it into another command, called a filter. The most common example of a pipe is more. For viewing a file, you just type "more <filename>". Suppose, however, that you use the w command, but there are more users than lines on your display. In order to read them all, you would type "w | more". The | character is called a pipe. It sends the output of the command on its left into the command on its right as the latter's input. The second process will start as soon as the first produces some output. Some other common filters are sort, which sorts the input it receives, and wc, which stands for word count. Word count can also be used to count characters and lines. See the man page for details.

Sometimes, however, you don't want to see the output on the screen, but save it in a file. To do this, you would redirect the output to a file (>). An example would be saving the current list of users to a file. To do this, you could type "users > users.list". This means of redirection can also be used to redirect the output from one process into another, but unlike the pipe, the second process would not start until the first one was finished.

Similarly, you can redirect the input of a process to read from a file (<). A good example of this is when you want to mail a text file to another user. To do this, you would type "mail -s "Good Joke" jsmith < funny". This command would send the file called funny to user jsmith with the subject line "Good Joke."

Pipes and redirection can be combined to execute complex tasks. For instance, suppose you want to get an alphabetical listing of the current users and what they're doing, making sure that the information doesn't scroll off the screen before you can read it. You could set up a series of pipes like this: "w -h | sort | more". If instead of viewing this information on the screen, you wanted to save it to a file, you could type "w -h | sort > whosdoingwhat".

You can also redirect to append the new output to a file (>>). Suppose you wanted to repeat the previous command, but not lose the old information. You would type "w -h | sort >> whosdoingwhat". The file would then contain both sets of information, one after the other.
Processes

UNIX allows users to have multiple processes running simultaneously. This is convenient when you have something to do that doesn't require any user interaction. In this case, you can have the process run in the background while you go on to do something else. To run a process in the background, simply type a space and an ampersand (&) after typing the command as you usually would.

It is important to note that if the process is going to produce screen output, it might interrupt whatever you are doing. The easiest way is to redirect the output (see the Redirection & Pipes section). For example, if you wanted to get a list of the remote users, but didn't want to sit and wait for the process, you could type "rusers > remote.users &". You would get a message saying the process had started, along with a process id. Then you can continue with whatever you wish and when the process is finished, you will get another message stating this. Then you can examine the remote.users file you created to view the output.

If you want to know what processes you have running, use the ps command. This command displays all significant processes you have running, there process id's and other information. To see all processes you have running, use the -a option (i.e. "ps -a").

If you decide you don't want a process to complete for some reason, you can kill it with the kill command. The format is "kill -9 <pid>", where <pid> is the process id. The -9 option ensures that the process will be killed, no matter what. Otherwise, the process might have some sort of error trapping to prevent it from being killed. No process can trap the kill -9. Be careful not to kill the shell you are running in. This is similar to logging out, but not as graceful. Note that if you do log out or kill the shell, all other processes will be lost.
.project File

This file resides in your home directory. It is displayed when someone fingers you. Only the first line of this file will be displayed. This line will appear immediately above your .plan file, if you have one. You must set the permissions so that others can read the file if you wish it to appear when you are fingered. (Use "chmod 644 .plan".) UNI does this for you automatically.
.plan File

This file resides in your home directory. It is the last thing displayed when someone fingers you. This file is not limited to any length, but it is a good idea to keep it short so that when someone fingers you, all the information will fit on one screen. You must set the permissions so that others can read the file if you wish it to appear when you are fingered. (Use "chmod 644 .plan".) UNI does this for you automatically.