

<b>USER GUIDE TO UNIX SYSTEMS</b>	2
<b>Read Me first</b>	2
Getting Help	2
<b>The Apply Procedure</b>	3
When to Run Apply	3
Choosing a Good Password	3
Logging In or Running Apply at a UNIX workstation	4
How to Apply (Running the Apply Procedure)	4
Show ID to Get Your New Account Approved	5
Top Reasons Your Account Will Not Be Approved	6
The tcsh Shell and Login Environment	6
<b>Logging In and Communicating from Home</b>	6
SSH2 Is Required For these Connections	7
File Transfers	7
<b>The MOST Basic UNIX Commands</b>	7
The ls Command	7
Example: Copying an Assignment to your Directory	8
Help in UNIX	9
The "man" command in UNIX	9
UNIX Reference Books	9
UNIX Command Shell	10
Modules and Paths	10
<b>Editors</b>	11
<b>The vi Editor</b>	11
vi Command Mode	11
vi Insert Mode	12
vi Last Line Command Mode	12
vi - More Advanced Commands	12
<b>Emacs</b>	12
<b>Mail</b>	13
The pine Email Command	13
Getting mail on your PC through the Internet	13
<b>The C Programming Environment</b>	13
The gcc Command: Compiling C Programs	14
The gdb Command: Debugging C Programs	15
The gdb Command - more advanced commands	16
The script command	17
UNIX file protection	18
<b>Who's Who in the CS Program</b>	19
<b>Map</b>	20
<b>Index To Guide</b>	21

# USER GUIDE TO UNIX SYSTEMS IN THE COMPUTER SCIENCE DEPARTMENT

## Version 13

August 27, 2006

### Read Me first

This GUIDE is intended as an aid for UNIX systems users at the Computer Science Lab of the Department of Computer Science at UMASS/Boston. This GUIDE was originally written for students in CS240, but we hope it will be useful to everyone.

If you are new to this and are taking a course in which UNIX will be used, the first thing you need to do is go to the **UNIX LAB** in S-3-158 (Science Building, Floor 3, Room 158.) See the **MAP** in this Guide for directions. Note that all bold terms in this Read Me First section can be looked up in the index for immediate reference. In the UNIX LAB you need to go through the **Apply Procedure**, explained in a section below. After a few hours, assuming the Apply Procedure worked, you should be able to perform the **Login Procedure**, explained in the section following the Apply Procedure. Various methods of performing **login from home** and executing **file exchanges** from your home PC are also explained in that section.

Following this, we deal with UNIX commands for file creation and modification (**Editors**), communication by **email** (probably the **pine** command), and program development (**gcc**, **gdb**, **script**, **file protection**, etc.). The Guide ends with a **Who's Who** giving instructors and staff in the department that you might wish to contact and a **MAP** of the departmental area, followed by an **Index** for this document containing page numbers of all these terms and more.

### Getting Help

If you need help with UNIX details, consultants are available in the UNIX LAB (S-3-158) or in the adjoining UNIX Operations room (S-3-157) to the left of the UNIX LAB (the phone is 287-6480). If one consultant is unable to answer your questions, try another; they have various areas of expertise. You can also use **email** (e.g.: **pine**) to ask for help: just address your message to "operator@cs.umb.edu". Other students in the UNIX LAB are often helpful as well if you ask them about simple early problems with the system. Phone numbers and UNIX email usernames of instructors and graders are in the **Who's Who** Phonelist at the back of the printed version of this GUIDE. You can also find other information about instructors and graders, such as schedule and office hours, by using the **finger** command, explained below.

There are help pages on the CS Department Website. Go to [www.cs.umb.edu](http://www.cs.umb.edu) and see a menu bar on the left. Under "Student Resources", useful topics are "Computer Labs", "Programming Help", and "FAQ". Under "FAQ", see "How do I log on to the CS Networks from Off-Site" for example.

For questions, requests and complaints about normal operations, mail "operator@cs.umb.edu". Operations are supervised by the System Administrator, William Perry, username wperry. System Administrators Rick Martin, rickm, and Leonard David, ldavid, are also helpful with complex problems or if Perry is not available. These email usernames are also in **Who's Who**.

If you have suggestions for improving this GUIDE, please send mail to "poneil@cs.umb.edu". Good luck with your Computer Science experience at UMass/Boston!

# The Apply Procedure

## When to Run Apply

The apply procedure is what you use to initialize your account, give your user name and password, and specify what course you are using the account for. You should go through the apply procedure every time you take a new CS course, even if you already have a UNIX account. You keep your home directory and username for the duration of your studies here, but the apply program will give you a course directory, add you to the class mailing list, increase your printer quota, and give you permission to use any other resources that you may need for each course. Some courses require a special server machine with unique capabilities (e.g., database use), and you should expect your instructor to tell you which one to use. Then after logging in, you can use the **rlogin** command to connect to that machine as your home; alternatively, you can use an Xterm Window to connect.

## Choosing a Good Password

Before applying, you need to choose a good password (we check how good your password is, and will fail your apply if it is not up to specification), so you need to understand what a good password looks like. A good password is at least six letters and is not in any dictionary or any list of people or place names. It combines numbers, upper and lowercase letters, and ideally contains at least one non-numeric, non-alphabetic symbol such as (~<>|#\$%^&\*).

Passwords are too easy to guess if they contain:

- any part of your name or name+initials or name+date
- any part of any name found in the password file
- names or words that are backwards
- words with mixed capitalization (NoTHarD)
- words with cute misspellings (WareZD00D)
- words with a single digit added (like "pascal1")
- words with substituting "1" for "i" or "0" for "o" (M1ll10n)
- two small words put together (badbad, baddab)
- strings that are all numbers (123321)
- short or repeating nonsense words ("glup" or "frgfrg")
- acronyms or scrambled words (umbcslab, aajv011)
- any systematic, well-adhered-to algorithm whatsoever (for example, suggestions patterned after something in a book)

One way to choose an OK password is to pick a phrase or song and use the first letters of the words, with some non-letter characters.

Cracking software comes with huge "dictionary" lists of words in many languages, real and imaginary, and English transliterations of languages in other alphabets: Russian, Mandarin, Swahili,

Vietnamese, Arabic, Farsi, Hindi; all the words in the Koran; the CIA World Factbook's list of every place name in the world; car names, people names, botanical names, science fiction names, Klingon words; and just about anything you could think of. If it has been written down anywhere in the world, it's probably in a cracking dictionary. Here are examples of bad passwords modified from the Security FAQ:

- alec7                   it's based on the users name (and it's too short anyway)
- gillian                girlfiends name (in a dictionary)
- naillig               ditto, backwards
- PORSCHE911        it's in a dictionary
- 12345678            it's in a dictionary (and people can watch you type it easily)
- qwertyuiop         or any substring, ditto with 12345678
- Computer            just because it's capitalized doesn't make it safe
- wombat9             ditto for appending some random character
- 6wombat             ditto for prepending some random character
- merde3              even for French words...
- mr.spock            it's in a sci-fi dictionary
- zeolite              it's in a geological dictionary

ANY password derived from ANY dictionary word (or personal information), modified in ANY way, constitutes a potentially guessable password.

## **Logging In or Running Apply at a UNIX workstation**

The UNIX workstation should be displaying a screen with a login and password prompt. Using the mouse, select the login field by placing the mouse pointer within the field and clicking the left mouse button. If the UNIX workstation screen is dark, clicking the mouse or touching any key on the keyboard should wake it up. If the workstation does not display a login screen, please try another workstation or ask the operators in room S-3-157 for assistance.

To log out from an UNIX workstation (after you have a password: Apply will log you out automatically), use the left mouse button to bring up the Root Menu and select Quit. To be certain, you may want to wait until the Login screen appears. You do not have to log out of your workstation windows first, but this is a good habit to make sure that you have saved any work in progress.

## **How to Apply (Running the Apply Procedure)**

UNIX distinguishes between lower case and upper case, so you need to consistently type all of your lines in lower case and end with a carriage return, unless directed otherwise.

To Apply, you need to take the following steps:

- Read the previous section on Choosing a Good Password
- Proceed to a UNIX workstation or PC in room S-3-158
- At the login: prompt, type `apply` and press RETURN.
- At the password: prompt, press RETURN.

- Follow the rest of the instructions Very Carefully.

**WARNING:** Do not use the arrow or the backspace key to erase your mistakes. Use the delete key or wait until the end of the process to correct them. You will be asked if you already have an account on cs, meaning any CS Department computer. If you are not sure if you have an account, see an operator in room S-3-157. If you have an account, please use your existing username.

If you do not already have an account, you will be asked to choose a username. Usernames can be between two and eight characters. Your username **MUST** be all lower case letters or lower case letters and numbers and should start with a letter.

We **strongly suggest** that you use the first letter of your first name, followed by the first 7 letters of your last name, that easily identifies you. For example: "jmcnama" for James McNamara. If someone else is already using your first choice you will see the password: prompt. This means you will have to choose another name. Our example James McNamara might use: "jmcnam2".

You will now be asked for your real name. Type it in using uppercase and lowercase letters in natural order, **NOTE:** "James Q. McNamara" not "McNamara, James Q."

You will be asked what password you want. Read the prior section in this Guide as to how to choose a good password. Passwords are checked periodically by a program that tries all the dictionary words and additional "easy" passwords such as "xyzy" or nicknames. Accounts that fail the password check will be disabled temporarily, so make up a good password at the beginning.

If you are applying for a new account, apply will ask you to choose a type of account: choose undergrad or grad. Apply will now ask you to select the name of the course and instructor you are applying for; type in a number of an item from the list, 1-18 (or so). You may choose more than one course. When you have no new courses to add, type " q" for " quit" in place of a number.

## **Show ID to Get Your New Account Approved**

If you have never had an account before on the system, go and show student identification to the Operator in the UNIX Operations Room (S-3-157). **ACCOUNTS ARE NEVER ACTIVATED UNTIL YOU SHOW AN ID TO THE OPERATOR!**

You should receive the account you asked for in a few hours. Test if the account exists by trying to login, following the same procedure you did to apply, except that in response to the "login:" prompt from the system you should respond with your username. You will then be prompted for your password, and after providing it should see a UNIX prompt ending in "%".. If you do not have an account after a few daytime hours have passed, go to the UNIX Operations Room (S-3-157) and find out why. Do not run apply again unless the operator asks you to.

If you get an account but it has problems, send mail to "operator" or go visit one.

**IMPORTANT:** Never share your password with ANYONE. We mean it. We have a strict one-account, one-user policy. We can help you communicate with other systems and share group files without sharing passwords. If you share your password you might lose your UNIX account.

## **Top Reasons Your Account Will Not Be Approved**

1. You Do Not Show an ID to an Operator in room 157 (**MOST** common)
2. Your username starts with a number or a capital letter
3. You currently have a disabled account on our system

4. You asked for a faculty, staff or courtesy account without approval
5. Your full name field is nonsense or indecipherable

As explained above, after your apply has been successful, you can approach any workstation in the UNIX LAB and duplicate what you did to apply. When the workstation prompts "login:" you should reply with your username. Then when it prompts for a password, give the password you chose and you should be successfully logged in (see a prompt from the system ending with "%"). See the section below on The Most Basic UNIX Commands at this point, since you should be in UNIX Command mode in the "tosh shell". The remaining subsections of this Section deal with details such as recognizing your shell (UNIX Command Environment), logging in from home, and transferring files from your home PC to the UNIX Server and back.

## The tosh Shell and Login Environment

When you login to your UNIX account, you should see a prompt that begins with the host name and contains the percent-sign, "%", like so:

```
blade41%
```

This is how you know you are in the tosh shell. (Actually, this is the behavior of the slightly simpler C shell too.) An incompletely initialized account may have the "\$" prompt instead. The tosh (pronounced "tee see shell" or "tee shell") is the shell (command environment) you are given when you apply for the first time. The shell is the part of UNIX that understands the commands you type. If you have the wrong environment, everything will act strangely and you may have other problems, so ask the operators in (S-3-157) for help.

NOTE: UNIX has more than one possible shell; two common ones are sh, the "Bourne Shell" (with a "\$" prompt) and csh, the "C Shell" (with a "%" prompt). Sun Solaris and AT&T System V machines will also have ksh, the "Korn Shell". A popular shell on machines with GNU software is bash, the "Bourne-Again Shell." The CS Lab has adopted the "tosh" shell as standard. The "tosh" does everything that the "csh" does, plus it has some useful extra features including emacs-like command editing.

If you are reading UNIX books, pay attention to what shell the book is using in its examples. Each shell is slightly different, so a program written for one shell often will not run on any of the other shells.

## Logging In and Communicating from Home

Most students have a PC at home and access our network via the Internet. There are two methods for doing this.

1. Get a PPP connection from an ISP (Internet Service Provider), connect to them then, use the Internet to access our systems. Look up Internet services in your local phonebook yellow pages or ask your local nerd.
2. Get a DSL or cable modem connection (about \$50/month). No phone line is tied up while you work, and you can directly connect to our systems over the Internet.

## SSH2 Is Required For these Connections

"SSH2" means "Secure Shell Protocol", version 2. Go to [www.cs.umb.edu](http://www.cs.umb.edu) & navigate the menu bar on the left to "Student Resources/FAQ/How do I log on to the CS Networks from Off-Site"

## File Transfers

Go to [www.cs.umb.edu](http://www.cs.umb.edu) & navigate the menu bar on the left to "Student Resources/FAQ/How do I transfer files to/from my PC"

## The MOST Basic UNIX Commands

These are the basic commands:

cat     print out a file to your workstation screen all in one stream (see "more" below)  
cd     change directory - move from one directory to another  
cp     copy a file, to a file with a new name and/or in a different directory  
login   how to get online; explained in article on the apply procedure  
lpr     print a hardcopy of a text file on the lineprinter  
ls     list files existing in a directory - many different options, explained below  
more    print out a file to your workstation screen one page at a time  
mv     move a file from one place to another; or change name of file  
mkdir   create a new subdirectory  
pwd     print working directory - list the "pathname" of the directory you're in  
rm     remove, that is, delete named file  
rmdir   remove named subdirectory

CTRL-C (hold down CTRL key and hit C) interrupt current ongoing action

You need all of the commands above to function at all. If you do not understand the idea of tree directory structure, read the overview book or get someone else to explain it to you.

## The ls Command

The `ls` command lists the names of files in the current directory or in a directory named as an argument. The `ls` command has many *options*, which are hyphenated letters following the command that modify the action it performs. For example, filenames beginning with a "period" are normally not listed, but such files will be listed if the "-a" option is used ("-a" stands for all files). You can see the size in bytes, last modification date, and file protection for each file with the "-l" option (for "format = long"). These options can be combined: `ls -la` or `ls -al`. The "man `ls`" command will show you a (numbingly complete) list of options for `ls`.

The `ls` command can also use wildcards, such as "\*" to stand for an arbitrary string in a character name. For example, to see only files whose names end with ".c", representing C source files, type:

```
blade41% ls *.c
```

The `ls` command only shows one directory at a time. To get a listing of all files ending with ".c" in any subdirectory of the one you are in, you can use the more powerful "find" command:

```
blade41% find . -name "*.c" -print
```

The initial "." means to search down from the current directory, the "-name" means the file name, the " character is placed before the \* to give the wildcard to find instead of it being interpreted by the shell, and the " -print" option at the end tells find to show you the results.

## **Example: Copying an Assignment to your Directory**

As an example of a skill you need, you will probably be asked to get copies of homework assignments from your instructor's directory. Let's say your instructor is Prof. Morris. You can find his username with the finger command:

```
blade41% finger Morris
```

The command will give you any information it has about a user by that name (think of it as an on-line "Who's Who"). If your instructor is Patrick O'Neil, O'Neil being a name with an embedded apostrophe ('), quotes are needed around the name:

```
blade41% finger "O'Neil"
```

The command will give information on Patrick O'Neil and Elizabeth O'Neil, since both users exist, as well as any others with the same names. The finger command will also give you some extra information which is in a file named ".plan" created by that user (poneil), for example containing schedule, office hours, and home phone number. You can create a ".plan" file in your own main directory, to give your home phone number, schedule, and whatever other information you want to make available.

Knowing your instructor's username, you can move to O'Neil's home directory by giving the command:

```
blade41% cd ~poneil
```

Type pwd to see that you have been successful. Do ls -CF to see what files and directories exist (the directories will have an attached "/" sign). You should see "cs240" as one of the directories. To move down into that subdirectory, you can simply type:

```
blade41% cd cs240
```

Now, pwd should show that you are in the directory /home/poneil/cs240. If you type ls again, you should see a subdirectory "hw1" and you can now type "cd hw1" to get down to that directory. Type ls again, and you should see a file named "assignment". You could have shortened the search above if you knew the subdirectory names in advance by typing:

```
blade41% cd ~poneil/cs240/hw1
```

Now you can type more assignment to list the assignment on your workstation screen. To copy this file over to your own directory, type:

```
blade41% cd
```

alone on a line to return to your home directory. Now type

```
blade41% cd cs240
```

to get into your special course subdirectory. If you do not have a course subdirectory, you need to run apply for cs240 for this semester and you will get one. Next type

```
blade41% mkdir hw1
```

to create your own hw1 directory. Now type "cd hw1" and copy the assignment over from poneil by giving the command:

```
blade41% cp ~poneil/cs240/hw1/assignment .
```

The SECOND argument of this cp command is a single period, "." meaning that the file should have the same name and be placed in the current directory. A full pathname could be used instead:

```
blade41% cp ~poneil/cs240/hw1/assignment ~yourname/cs240/hw1/
```

Always be careful that you are in the right directory when you copy something into your current directory using the "." argument; copying to the wrong directory is a common beginner's error. You can always check where you are with the "pwd" command.

## Help in UNIX

UNIX does not come with a built-in help command. Instead, the man command displays pages from the UNIX manuals.

Our local help is available on the web by connecting to our main page, <http://www.cs.umb.edu>, and then clicking on Resources. We **strongly suggest** that you read the help sections on Frequently Asked Questions (FAQ).

## The "man" command in UNIX

The man command comes with the UNIX system and provides descriptions of UNIX commands in full detail. The detail can sometimes be quite overwhelming, but there is a lot of useful information in these pages.

## UNIX Reference Books

You definitely need to buy some sort of UNIX reference book. It helps, when choosing a book, to know that there are historically two "flavors" of UNIX: AT&T's System V, "SYSV" for short, and Berkeley's BSD. Most of our user machines run Solaris 2.6, which is a mixture of the two but somewhat more System V-like; a few of our machines run SunOS 4.1.4 which is a BSD type UNIX. The SAPC lab, used by CS241 and some sections of CS444, has PCs running Linux which is another hybrid UNIX with freely available source code.

The CS240 class is using the UNIX text "UNIX for Programmers and Users", [The Latest Edition] by Graham Glass King Ables, Prentice Hall, referred to in what follows as **Glass**.

## UNIX Command Shell

In the tcsh, the default UNIX command shell, there are two files in your directory which control your environment: ".login" and ".cshrc". These files will not show up when you type ls alone, because files whose names begin with a period are normally invisible. To see that they are there, you need to type:

```
blade41% ls -a
```

You can edit these files to make changes to your options. The `.login` file has a section that is commented as being too technical for a beginner. Prove us wrong if you want, but be warned.

If you make changes to these files and have problems, you can copy the original "new user" version into your home directory, but you will also wipe out any useful changes you have made. To perform the copy, give the following sequence of three commands:

```
blade41% cd (get to your login directory)
blade41% cp /usr/local/lib/.login .
(the second argument above and below, "." (period), means current directory)
blade41% cp /usr/local/lib/.cshrc .
```

## Modules and Paths

In UNIX, commands can live in many different directories. Your "path" is an "environment variable" which tells UNIX which directories to search for commands.

We use a package called Modules which allows you to set your path and environment variables using "modulefiles."

Each modulefile contains the information needed to configure the shell for one application. Your instructor may tell you to edit your `.cshrc` file and add a module for your course. Typically modulefiles instruct the module command to alter or set shell environment variables such as `PATH`, `MANPATH`, etc. Modulefiles may be shared by many users on a system and users may make their own collection to supplement or replace the shared modulefiles.

The command

`module avail`

will show you a list of installed modules. See `man module` for additional information on module files. If you are accessing a database in your course, for example, you might need to check that your `PATH` is set correctly.

NOTE: Make sure that needed environment variables are set in your `.cshrc`, not in your `.login`. Your `.cshrc` is executed every time you start a shell (including X workstation windows, sub-shells, shell commands from an editor, etc.; ) but your `.login` is only executed once, at login, after the `.cshrc`. This means that anything you set in your `.login` will not be set for sub-shells. Use your `.login` to set your terminal type and do house-keeping, and put everything else in your `.cshrc` file.

IT IS A VERY GOOD IDEA to take a look at the default `.login`, `.cshrc`, and `.xsession` files in `/usr/local/lib` every year, and compare them to yours.

## Editors

There are two important editors in use on the UNIX System. These are `vi` and `emacs`. Additionally, there is the `ed` editor, a very simple line editor, which does not require full-screen capability. This was once useful for printing paper terminals, but the only reason to know it now is that `vi` is based on `ed` and some manual pages refer to it.

The `emacs` editor is the most powerful one, but rather complex to master, and not available at many sites. Some instructors recommend `vi` for new users, and others require `emacs`.

# The vi Editor

The vi editor has a tremendous number of commands (too many) This article explains only the basic commands. There's also a list of less frequently used commands, but this is still only a small fraction of the total that exist. Don't worry though; most people who use vi never learn most of the commands and never miss them.

You can create and edit a file with the name "fname" by typing:

```
blade41% vi fname
```

If the file "fname" does not already exist, an empty file with that name will be created. The vi editor has three major modes of operation: command mode, insert mode and last line command mode. You enter the vi editor in command mode, and your screen will show a window on the first page of existing text in the file.

## vi Command Mode

In command mode, you can move the cursor around using the "arrow" keys, or perform any of the following actions by typing the character string commands **without typing a return** (you will not see the characters, only the action):

command	action
[n]x	delete the [next n] character[s] under the cursor
[n]dd	delete the [next n] line[s] under the cursor
[n]G or :n	move cursor to line number n
G	move cursor to last line of the file
i	enter insert mode, before character under the cursor (need "i" to insert at beginning of line)
a	enter insert mode, after character under the cursor (need "a" to insert at the end of line)
o	enter insert mode on NEW LINE created below current line (need "o" to insert new last line in file)
O	enter insert mode on NEW LINE created above current line (need "O" for the first line in file)
:	(colon :) enter last line command mode
ZZ	save file and return to UNIX

Some of the commands may be preceded by an integer, symbolized by "[n]" in the above, which repeats the command action "[n]" number of times. Thus, typing x three times deletes the next three characters, or you can type 3x to do the same thing with a single command. The command 3dd will delete three lines, starting at the current line.

**vi Insert Mode.** In insert mode, everything you type will be placed in the text. To leave insert mode and return to command mode, hit the ESC key.

**vi Last Line Command Mode.** The colon character, ":", will bring you from command mode to last line command mode. The colon will appear at the beginning of the last line along with successive character you type, and you should terminate these commands with a RETURN. Historically, last line mode is equivalent to the old ed editor, and you may see these called "ed" commands:

command	action
:w filename	write contents of file to file "filename"
:w	write to default filename the was opened by vi
:r filename	read file "filename" into current buffer
:q (or :q!)	quit back to UNIX without saving updates to file

The RETURN at the end of line returns you to the normal command mode.

**vi - More Advanced Commands.** This is still only a subset of the vi editor's commands.

command	action
<b>/STRING</b>	<b>SEARCH FORWARD IN TEXT TO FIND "STRING"</b>
?string	search backward in text to find "string"
J	Join the next line with the current one --i.e. delete intervening newline char
[n]yy	yank (cut out copy) of next n lines
p	paste yanked lines into text on new lines following current cursor
D	Delete characters to end of current line

There is also an "ed" command to change text, which can be used by vi, like all ed commands, by prefacing a colon to enter last-line command mode; this is particularly useful for global edits:

```
:m,n s/text1/text2/g
```

This will search from line m to line n and change all occurrences of text1 strings to text2.

## Emacs

Emacs is more than an editor; it is a flexible, extensible command environment. However, it is not available at all UNIX sites so being able to use the vi editor is a valuable skill. Where Emacs is available -- for example at UMass/Boston -- some users spend all of their time online in Emacs. Emacs has extensions to compile and debug programs, edit directories and manage files, send mail, read Usenet news, and even act as a psychoanalyst. Emacs comes with its own LISP language.

Emacs also has a tutorial, and the best way to begin using Emacs is to start Emacs and run the tutorial. Enter emacs and then type CTRL-h-t (Hold down the Control key and type CTRL-h, then release the control key and type t.) For a full listing of emacs command see the URL

<http://www.cs.umb.edu/helproot/emacs/refcard.html>

# Email

The default mail program is the "BSD" or "ucb" mail program, covered below. This is a simple, command-line program that does NOT handle attachments. You will find this mail program on every UNIX system, but there are a number of newer mail handling programs. Two that we support here are **pine** and **elm** (**pine** is the one that is more commonly used) called by typing these names as commands at the system prompt.

The programs pine and elm are very similar to each other, and both have on-line help. elm has a more sophisticated ability to sort and filter messages; pine handles more complex (MIME) types of mail with ease. In particular if you receive email with attachments, you can use pine to forward your mail to a PC email address that can deal with this better.

## The pine Email Command

Pine is the system to use when you receive mail with what seems to be gibberish added at the end, especially if the term "MIME" appears before the gibberish. To read your mail using the pine mail program, give the UNIX command

```
blade41% pine
```

After you give this command, you will be placed in an environment with menus of single-letter commands. The simplest approach for dealing with a MIME message is to forward the message through pine to your home PC username. From there you should be able to read the attachment and interpret the various formats: .pdf (through Acrobat reader), .doc (through Microsoft Word), etc.

## Getting mail on your PC through the Internet

If you have a home PC with a PPP connection to the Internet, you can retrieve your mail from our mail machine using Netscape, Eudora, or any other PC mail client. See the IMAP/POP topics in FAQ section of USER HELP. Our pop server is at pop.cs.umb.edu and the smtp server is smtp.cs.umb.edu. Please use these aliases as they are guaranteed to work.

## Forwarding UMB Email

You can forward your email from our site to your preferred email address. See "How to forward my \*@cs.umb.edu account email" in the FAQ under "Student Resources" on the UMB Department Web Page Menu (on the left).

# The C Programming Environment

The first course at UMass/Boston dealing with the UNIX System and C Programming is CS240. The basic C texts are The C Programming Language, 2nd Edition, by Kernighan and Ritchie (abbreviated "K&R"), and UNIX for Programmers and Users, Latest Edition, by Glass and Ables (Referred to as Glass). There are a number of procedures you will need to master for CS240 that we cover here, some of which are non-standard in the UNIX world. You will use the gcc command to compile your program, and the gdb debugger to run it under interactive control to find errors in logic (debug the program). You will use the script command to prepare a turn-in that demonstrates the workings of your program assignments.

## The gcc Command: Compiling C Programs

The command `gcc` is the version of the `cc` command that we use in all programming classes; `gcc` is "Gnu cc", part of the "Gnu" software from the Free Software Foundation; similarly `gdb` is the Gnu debugger. To compile a C program, give it a filename ending in ".c", for example "prog.c" then compile it by typing:

```
blade41% gcc prog.c
```

This creates an executable file known as `a.out`. You can see this in your directory using the `ls` command, but don't try to edit "a.out", since the format is not one that can be viewed. If you type `a.out` alone on a line, followed by a carriage return, you will execute the program. If you want to give a specific name other than `a.out` to your executable file, say the name "runfile", you can compile with the `-o` flag:

```
blade41% gcc prog.c -o runfile
```

and now typing `runfile` alone on a line will execute the result. To be able to debug your program (as explained in the `\index{gdb}` article below), you would use the flag `-g`.

```
blade41% gcc -g prog.c -o prog
```

**HINT:** Do NOT name your program "test.c" or make an executable program called "test!" The UNIX shell contains a "built-in" command called `test` and this will run instead of your program!

As you will learn in Chapter 4 of K&R, it is possible to have different routines of a single program in different files. The different files might be named as in the example of Chapter 4: `main.c`, `getop.c`, `stack.c` and `getch.c`. Then you can compile all of these files at once into an executable file named `calc`, allowing for debugging use, by giving the command:

```
blade41% gcc -g main.c getop.c stack.c getch.c -o calc
```

This command will create "object" files in your directory, `main.o`, `getop.o`, `stack.o` and `getch.o`. These are files for which most of the compilation work has been done, but the final linked load into the executable is yet to be performed. Now if you need to modify a single one of these files to fix a bug, say `stack.c`, you can save a lot of time in recompiling by giving the command:

```
blade41% gcc -g main.o getop.o stack.c getch.o -o calc
```

By specifying the ".o" extension (suffix) on three out of the four modules named, the compilation step is skipped, only one module is compiled, and the result is link loaded together with the existing object files into `calc`.

In `cs240` you will learn about the `make` command, which is a program that manages compiler options and program building. **NOTE:** It is good practice to ask `gcc` for more warnings than it gives by default.

```
blade41% gcc -g -Wall prog.c -o prog
```

The `Wall` flag will give all (or almost all) warnings as well as errors. The makefiles for `CS310` give more examples of options.

## The gdb Command: Debugging C Programs

The gdb command lets you run a program so as to interactively control its progress and watch the results of its operations; in this way you will be able to detect how bugs arise. The first step in debugging a program is to compile it with the `-g` flag, as explained in the gcc command above:

```
blade41% gcc -g prog.c -o prog
```

Following this, you can give the command

```
gdb prog
```

The program is now loaded into the gdb environment, but it is not yet running. Before starting it running, you will probably first wish to set up a "breakpoint", to make it stop when it reaches a particular source line. As an example, you can set up a breakpoint to make the program stop after it enters `main()` (a number of things can happen in running a program before the `main()` function you wrote is entered), and then start it running, taking its standard input from some file named `infile`:

```
(gdb) break main
(gdb) run < infile
```

To reduce confusion you will want to provide interactive input to the program through an `infile`, so that all interaction during debugging is with gdb. After a breakpoint is reached, gdb will print out the C language statement reached. You can now make the program progress through single steps, executing consecutive source lines of program logic in the flow of control, by giving the command:

```
(gdb) s          -- s is for "step" -- enter and step through statements of function when
                  a function call is encountered
(gdb) n          -- n is for "next" - step, but skip over function calls
(gdb) r          -- run program
(gdb) r small    -- run program, and give it "small" as an argument.
(gdb) r < infile -- run program with input from file "infile".
```

As each step is executed, the corresponding source line will be printed out. You can type `CR` on successive lines after the first command to perform successive steps. At any time where your input is expected, you can print out the value of any variable in the scope of the current logic or an expression involving such variables (such as `3*i`), by typing:

```
(gdb) p 3*I     -- p is for "print"; i is a variable in scope of current position
```

The type printed will be the type derived from the variables in the expression; if special output formats are needed, you can use the `"p/format"` form for print:

```
(gdb) p/x 3*I  -- x for hexadecimal, o for octal, d for decimal, f for float,
                  c for char, s for string
```

You can also get the (default type) values of all variables in your scope by typing:

```
(gdb) i lo      -- i is for "info" -- gives values of local variables and current stack level
(gdb) i var     -- values of global and static variables
```

You can get relatively useful help messages while in the debugger by typing:

(gdb) h -- h is for help -- list of help topics  
(gdb) h topic -- help on named topic  
(gdb) h p -- help on print command (or any other command)

At any time, you can quit:

(gdb) q - q is for "quit", i.e. Leave the debugger.

## The gdb Command - more advanced commands

So far we have only told you how to set a breakpoint at the entry to the main program, and how to perform single steps in running the program. This is enough control only for very short programs. More generally, you need to be able to set breakpoints at arbitrary positions and continue running the program after one breakpoint until you get to another breakpoint.

(gdb) b 36 - break at line number 36 of current source file  
(gdb) b 36 if i==3 - break at line 36 of current source file if the variable i is equal to 3  
(A condition such as if "i==3" can be added after any breakpoint definition.)  
(gdb) b fn.c:22 - break at line number 22 of source file fn.c in this executable  
(gdb) b func - break at function "func" entry point (or in source file fn.c: b fn.c:func)  
(gdb) i b - info breakpoints. Lists all breakpoints now set up.  
(gdb) d - delete all breakpoints (d 1 for just #1).

An executable file (a.out if the -o option is not used) can be compiled from a number of source files in C, and the will know about these source files and the lines they contain. To find source line numbers so you can set breakpoints, use the following commands:

(gdb) l 23 - list to workstation screen 10 lines of current source centered at line 23  
(gdb) l fn.c:22 - print 10 lines from source file in fn.c centered at line 22  
(gdb) l func - print 10 lines around function "func" entry point (or l fn.c:func)  
(gdb) l - print 10 or more lines after lines last printed  
(gdb) l fn.c:1 - print all lines starting from line 1 in source file fn.fc

Some other commands for examining memory and program information are:

(gdb) i sources - info on sources - print the names of all source files  
(gdb) i lo - info locals: values of all local vars, current function  
(gdb) i var - info variables: values of global/static vars  
(gdb) i s - info stack:calls made to get to this execution point.  
(gdb) bt - same as i s  
(gdb) where - same as i s  
(gdb) up - go up one stack level (to caller)  
(gdb) down - go down one stack level (to called function)  
(gdb) x 0x20034 - examine memory address 0x20034  
(gdb) x/s 0x20034 - examine memory, addr 0x20034, as a string (also x/f)  
(gdb) display x - prints x each time program stops  
(gdb) whatis x - prints type information for x

NOTE: While running, CTRL-C brings you back to the (gdb) prompt to examine the program state. This is useful for programs that are in an infinite loop or hung and for debugging performance problems.

To begin running a program again after a break (including the first break at main) type:

(gdb) c - continue running program from stopped point  
(gdb) c 22 - continue, don't stop at this breakpoint again until it is encountered 22 times

In setting a breakpoint we were allowed to give a condition (b 36 if i==3). In a counted continue (c 22), the condition is not checked until the breakpoint has been encountered 22 times. To find out what breakpoints exist and delete them:

(gdb) i b - info on breakpoints - returns list of breaknumbers  
(gdb) d 3 - delete break number 3  
(gdb) d - delete all breakpoints

You can create an array starting at any position to print out values all with the same type:

(gdb) p array[3]@12 - will print out values starting at array position 3 for 12 positions

You can also call your own function from within the debugger:

(gdb) p my\_function () - calls a function  
(gdb) call my\_function () - same as p  
(gdb) p my\_function (10, "Boston") - call function with values  
(gdb) p my\_function (x, p->name) - call using variable values

Other gdb capabilities exist which are useful for heavy-duty debugging. There are some photocopied GDB Manuals (stapled, marked: "PLEASE DO NOT REMOVE FROM UNIX ROOM") either in the CS Department UNIX LAB or in the CS Library in room 157.

## The script command

The script command is used to make a record of an interactive session. If you type:

```
blade41% script      (begins recording in file "typescript")
. . .                (any sequence of commands)
blade41% exit        (ends script command, closes "typescript" file)
```

you will create a file named "typescript" which contains all the keystrokes you type on the keyboard and all the characters which appear on your workstation screen. Your instructor may ask you to create a script file to demonstrate your work. Commonly, you will be asked to list the contents of your program (use the cat command), perform compilation, and exercise the program to demonstrate that it works properly. The typescript file can be left in your directory or can be printed out to be turned in at class.

Be careful that if you have a typescript file you don't wish to lose that you rename it (mv command) before giving the script command again, since the new script command will wipe out the contents of the old typescript file.

Also DO NOT try to print out binary files (a.out or \*.o files!) The printer cannot handle binary files and you will create a huge mess.

## UNIX file protection

When you are given a new account, you will start off with settings that make any file you create readable by everyone else on the system. This is good because it lets you exchange information easily with other users, and learn from them. However, your instructor will want to place restric-

tions on sharing assignment work, so new users will find they have a subdirectory for each course they have applied for, cs240 for CS240, with special access control. Any file you create in that directory will be protected from access by normal users ("others" in the file protection nomenclature), although it will still be accessible to your instructor and grader (people in your grading group).

The file protection settings used in UNIX are explained in any UNIX reference manual. The command `ls -lg` shows file protection and group ownership. The `chmod` command allows you to set protection on individual files or directories. There are three categories of readers: `u` for "user", the original creator of the file, `g` for a member of a "group" with the user, and `o` for "other". Your instructor and grader are members of your group, everyone else is other. You will be able, using the `chmod` command to "add" (+) and "subtract" (-) capabilities for members of these user categories to have various kinds of access to your file. the types of access are: `r` for "read", `w` for "write" (or "update" as in an editor) and `x` for "execute". To disallow other students from reading and executing your file, `fname`, type:

```
blade41% chmod o-rx fname
```

while to allow such access use "`chmod orx fname+`". Notice that the standard meanings for these types of access change when they are applied to directories; consider giving the command which denies "x" access for the user category "o" on your directory, `fname`:

```
blade41% chmod o-x fname
```

The effect will be that other users are not allowed to execute a listing of your directory structure, and therefore all files existing in that directory are protected from access (even if the files are readable by "other", such users can't access them since they can't interpret the directory they sit in). This is the type of protection you start with in your "cs240" directory.

In your course directory the protections have been set for you. It is very important that you do not touch the protection on any files in the course directory. For complex reasons, if you change the protection you will not be able to correct it, and your grader will be unable to access your files.

(The reason, for the record, is that your course directories have a group ownership of your grader's group, to allow the grader to read your files, and the directory has the "setgid bit" set. The setgid bit makes new files and directories inherit the grader's group instead of \*your\* group. You are \*not\* a member of the grader group, so that only the grader and the professor can read everyone's files. If you touch the directory protection in any way, UNIX will not let you write this setgid bit since you are not in the group. The setgid bit will not be reset, and new files will NOT be readable by the grader, which means that your homework will not be readable by the grader.

If you understand this, you are not a beginner!)

## Who's Who in the CS Program

Note that more information is often available through the command "finger username". Office extensions are available within UMB by dialing the last five digits, starting with 7, e.g. 76444 for E. Bolker; from outside UMB, the area code is 617.

Full Name	Office Ext.	Office	Username	Home Phone (<Until)
-----------	-------------	--------	----------	---------------------

### Faculty

Ethan Bolker	287-6444	S-3-189	eb	617-969-2892 < 9 PM !!
William Campbell	287-6449	S-3-183	bwrc	617-547-2738 < 9 PM !!
Ron Cheung	287-6474	S-3-075	cheungr	
Robert Cohen	287-6452	S-3-070	rfc	
Peter Fejer (Chair)	287-6463	S-3-184	fejer	
Colin Godfrey	287-6479	S-3-088	cgodfrey	
Robert Morris	287-6466	S-3-176	ram	
Elizabeth O'Neil	287-6455	S-3-169	eoneil	617-354-6460 < 11 PM
Patrick O'Neil	287-6468	S-3-167	poneil	617-354-6460 < 11 PM
Kenneth Newman	287-6467	S-3-171	kwn	
Karl Offner	(work)603-884-1468		offner	978-443-4697 < 10 PM
Marc Pomplun	287-6443	S-3-92A	marc	
Dan Simovici	287-6452	S-3-173	dsim	617-731-3297 < 9:30 PM
Gabriel Spitz	287-6479	S-3-088	gspitz	
Jun Suzuki	287-6462	S-3-168	jxs	
Robert Wilson	287-6478	S-3-071	bobw	

### System Staff

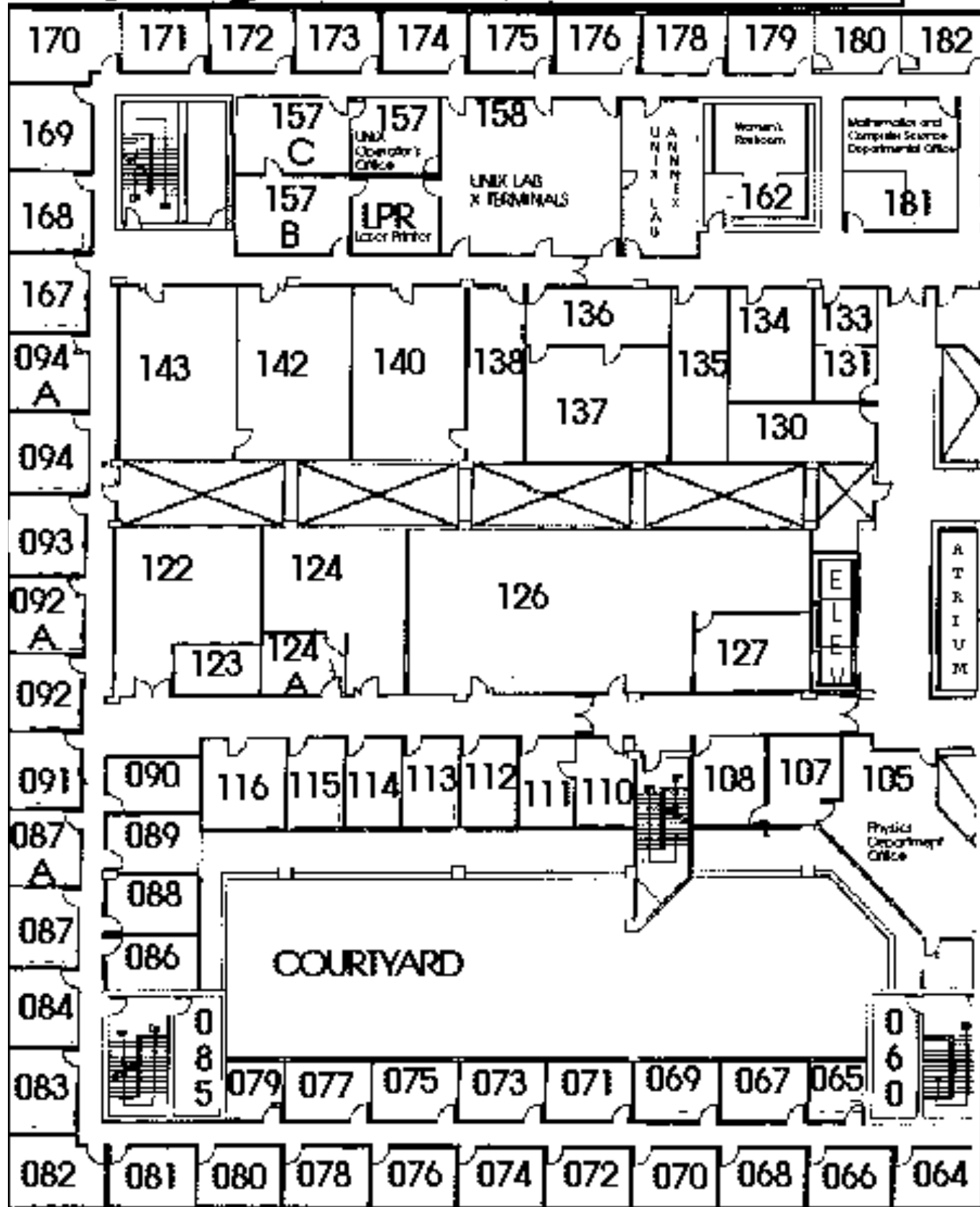
Leonard David	287-6477	S-3-072	ldavid	
Richard Martin	287-6465	S-3-089	rickm	
William Perry	287-6447	S-3-157B		wperry
UNIX Operator	287-6480	S-3-157	operator,	consult, accounts

### Office Support

CS Office	287-6440	S-3-132		
Karen Means	287-6441	S-3-132	karen	
Maureen Title	287-6476	S-3-132	mtitle	

University of Massachusetts, Boston  
 Department of Mathematics and Computer Science  
 Science Building, Third Floor

183  
 184



# Index To Guide

.cshrc .....	10
.login file .....	10
apply procedure .....	4
each new course .....	3
Failure .....	6
Bourne shell .....	6
C shell .....	6
compiler, gcc .....	14
warning flags .....	15
debugging with gdb .....	15
editor	
vi .....	11
emacs .....	13
extension (suffix for a file) .....	14
file protection .....	18
gcc compiler .....	14
gdb debugger .....	15
Glass text .....	9, 13
gnu software .....	14
home login .....	6
Kernighan and Ritchie text .....	13
logging in from home .....	6
make command .....	14
MAP .....	20
password	
choosing .....	3
pine email .....	13
protection of files in UNIX .....	18
script command .....	17
shell	
Bourne .....	6
tcsh .....	6
tcsh shell .....	6
typescript file .....	17
UNIX file protection .....	18
username	
choice .....	5
vi	
commands .....	11
cut and paste .....	12
last line command mode .....	12
search .....	12
vi editor .....	11
Who's Who List .....	19