

---

---

# Getting Started with Unix.

1997-98

RAHUL SIMHA

*Department of Computer Science  
The George Washington University  
Washington, DC 20052*

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Basic Unix</b>	<b>4</b>
2.1	Introduction to Unix . . . . .	4
2.2	The File System . . . . .	6
2.3	More Unix and System Information . . . . .	8
<b>3</b>	<b>Default Files</b>	<b>10</b>
3.1	The .login File . . . . .	10
3.2	The .cshrc File . . . . .	11
<b>4</b>	<b>Editing: the Emacs editor</b>	<b>15</b>
4.1	Getting Started . . . . .	15
4.2	Saving and Exiting . . . . .	16
4.3	Buffers . . . . .	16
4.4	Deleting and Undoing . . . . .	16
4.5	Moving Blocks . . . . .	17
4.6	More on Emacs . . . . .	18
<b>5</b>	<b>Creating Your Own Web Page</b>	<b>19</b>
<b>6</b>	<b>L<sup>A</sup>T<sub>E</sub>X</b>	<b>20</b>
6.1	Getting Started With L <sup>A</sup> T <sub>E</sub> X . . . . .	20
6.2	A More Complicated Example . . . . .	23
6.3	Understanding the Input File . . . . .	26
6.3.1	Common Commands . . . . .	26
6.3.2	Environments . . . . .	28
6.4	Math Mode . . . . .	29
6.4.1	Math Environments . . . . .	29
6.4.2	Common structures . . . . .	30
6.4.3	Arrays . . . . .	31

6.4.4	Multiline Formulas . . . . .	32
6.4.5	Mathematical Symbols . . . . .	32
6.5	Doing More With L <sup>A</sup> T <sub>E</sub> X . . . . .	32
6.5.1	The tabular Environment . . . . .	32
6.5.2	Tables and Figures . . . . .	33
6.6	Preparing a Document . . . . .	33
6.6.1	Sectioning Commands . . . . .	34
6.6.2	Table of Contents . . . . .	34
6.6.3	The Title Page and Abstract . . . . .	34
6.7	Error Handling . . . . .	35
6.8	Formatting, Previewing and Printing . . . . .	36
6.9	Other Things to Know . . . . .	36
<b>7</b>	<b>Drawing Pictures and Plotting Graphs</b>	<b>37</b>
7.1	Drawing using idraw . . . . .	37
7.2	Plotting graphs using gnuplot . . . . .	38
7.2.1	Introduction to gnuplot . . . . .	38
7.2.2	plotting $(x, y)$ pairs . . . . .	38
7.2.3	Additional Features . . . . .	39
<b>A</b>	<b>Unix Commands</b>	<b>44</b>
<b>B</b>	<b>Key bindings</b>	<b>47</b>
<b>C</b>	<b>Preamble for L<sup>A</sup>T<sub>E</sub>X</b>	<b>48</b>

# Chapter 1

## Introduction

This handbook will help you get started with: Unix, setting up default files, the Emacs editor, the L<sup>A</sup>T<sub>E</sub>X<sup>1</sup> typesetter, the idraw drawing package and the GNU-plot plotting package. All attempts have been made to make this guide as readable and uncluttered as possible. As a result several important concepts have been left out. While this guide is no substitute for the relevant manuals, it tries to get you started right away without too many hassles.

---

<sup>1</sup>This User Guide was written using L<sup>A</sup>T<sub>E</sub>X

# Chapter 2

## Basic Unix

### 2.1 Introduction to Unix

This manual is written under the assumption that you have little or no familiarity with Unix.

This manual assumes you are going to use `tcsh` as your shell. What is a shell? This is the command-line interpreter akin to the DOS prompt in DOS. In Unix, you have a choice of command-line interpreters and usually your system will pick one as the standard. You can change to another quite easily. If you are running `ksh` instead, you can run `tcsh` by simply typing `tcsh`. Note: `ksh` uses a different set of initializing files – this manual does not cover `ksh`. *Most `ksh` and `tcsh` commands are identical.*

When you log on, Unix places you in your “home” directory. Now type `ls -a` to see the names of the files in your home directory. If you have never used Unix before, you should note that `ls` is a Unix command which gives a directory listing (basically the same thing as “`dir`” in MS-DOS). The `-a` is an option, or a switch associated with `ls`. We will see more on that later.

The files initially in your home directory include `.cshrc` and `.login`:

```
ickis% ls -a
./      .login*
../     .cshrc*
```

Let’s look at the contents of one of the files. Type<sup>1</sup> `ickis% more .cshrc` Here’s what you should see (something like this, at any rate):

```
# @(#)Cshrc 1.1 86/07/09 SMI
#####
#
#      .cshrc file
#
#      initial setups for both interactive and noninteractive
#      C-Shells
#
#####

#      set up search path

set lpath = ( ) # add directories for local commands
set path = (/usr/local /usr/ucb /usr/bin /usr/bin/X11 /bin /usr/local/bin . ~ ~b
in)
```

---

<sup>1</sup> `ickis%` is the prompt, where `ickis` is the machine name. You do not type that in.

```

set path = ($path $1path /usr/games /usr2/public/bin /usr/local/bin/X11)
setenv MANPATH /usr/man:/usr2/public/man
setenv EMACSLOADPATH ./usr/local/emacs/lisp
setenv TEXEDIT "/usr/local/bin/emacs -l tex-start -e startline %d %s"
setenv TEXINPUTS ./usr/local/tex/macros
setenv XENVIRONMENT $home/.Xdefaults

#         cd path

--More-- (26%)

```

Hit the `<<Return>>`key to look at the rest of the file a line at a time or the space-bar to move through the file a page at a time. The contents of the file might look strange to you now, but we will discuss this file later in the manual. The important thing now is that you have just learned the `more` command to page through a file.

The `cat` command could also be used (try it now; at the prompt, type `cat .cshrc`), but it will not display the file a page at a time. Instead, you use the control characters `<<ctrl-s>>` and `<<ctrl-q>>` to pause and continue.

Other useful control characters are `<<ctrl-c>>` and `<<ctrl-d>>`, the “panic buttons”. Use `<<ctrl-c>>` to quit (cancel) any program or command and `<<ctrl-d>>` to halt execution. For example, do a `more .cshrc`, then instead of hitting the `<<Return>>`key or the space-bar, do a `<<ctrl-c>>` - you’re back at the command line (`<<ctrl-d>>` is treated like any other key). You may also find `<<ctrl-u>>` useful to wipe out what you’ve just typed out on the command line, e.g., `more .cshrc<<ctrl-u>>` will delete the entire command.

We have already stated that this manual is not exhaustive and cannot teach you everything about Unix and Xwindows. Therefore, you need to know how to get on-line help. On-line help in Unix is quite useful *if you already know the exact name of the command*. It is also the final authority for what the commands do on your system, since there is no standard Unix manual (Sun provides a manual for its systems). The help command really is a manual and you access that manual using the `man` command.

Try doing `man more`. You should get a ‘manual entry’ on the `more` command. Once again, hit `<<Return>>`to proceed a line at a time or any other key to move to the next screenful. (Note how it seems like you are using `more` to scroll through the manual. The `man` command uses `more`, through *pipes*, a mechanism for directing the output of one program to input of another, a Unix specialty for the advanced user. We will not discuss pipes here).

Proceeding with `man more`, under the heading of OPTIONS, we see a number of letters with *minus* signs before them. This is the Unix way of specifying options to a command. For example, do a `man 1s` to get help on `1s`, the directory command. You notice a bewildering number of options, most of which you will never use. Okay, do an `1s` followed by a `1s -r`. The ‘minus r’ option lists the directory in reverse alphabetical order. The ‘minus a’ option you used before lists all files including those that start with ‘.’ (such as `.cshrc`).

The `man` command is often the only way to get information about a particular command. Typically the “man pages” provide sufficient information on used, though that information may seem a little terse at first. However, if you did not know that the command for directory information is `1s`, then `man` is not much help. Typing `man directory` gets you nothing and `man dir` gives you information on a different command (`dir`). If you do not know the exact name of the command for which you want more information, then try `apropos` or `man -k`. For example, type `apropos directory`. Lots of Unix commands related to directories will scroll across the screen.

If `apropos` does not help, you simply have to rely on the available manuals and the people around you. Sometimes, if you know a related command, then clues might be found in the `man` entry for that command. For example, one of our debuggers is called `dbx`. Typing `man debugger` does not help, but if you already knew that the C compiler is called `gcc` then by doing a `man gcc` and looking under the SEE ALSO heading, you get some clues (c’mon, try it). Some useful Unix commands will be covered later in the manual and also appear in Appendix A.

## 2.2 The File System

The Unix file system is organized into the usual tree structure of directories and subdirectories. Your home directory is your username (`/home/mom2/yourname`). Create a subdirectory called `silly` by typing `mkdir silly`. (The command `mkdir file` creates a new directory with the name specified by `file`.) Now type `ls -F` to see that a subdirectory is listed as a file with a frontslash `'/'` following the name of the subdirectory, in this case, `silly/`.

To change directories, use `cd`, as in `cd silly`. This places you in the `silly` subdirectory. Now typing `ls` should show no files in the `silly` directory. Let's create one by typing `cp ~jones/.cshrc junk`. The `.cshrc` file in your home directory has been copied into a file called `junk` in your subdirectory `silly`.

To return to your home directory, type `cd`. The file `junk` in subdirectory `silly` can be accessed from your home directory. For example try `more silly/junk`. So you see that the file name is last in a sequence of names separated by slashes; the rest are directory names. Also, the home directory is always specified with a tilde `'~'`. Thus, the path to the file `junk` is `~jones/silly/junk`; this can be shortened to `~/silly/junk`. Here are some useful file commands with examples:

`cd ..` – move up from a subdirectory to a parent directory

`mv` – move or rename files, e.g., `mv junk junk2`

`pwd` – print the name of the current directory

`rm` – delete a file, e.g. `rm junk`

`rmdir` – delete a directory, e.g. `rmdir silly` (when you are in its parent directory).

Try these commands out with the subdirectory `silly` and file `junk` that you just created. Also, note that if you do a `man` on one of these commands (for instance `man rm`), you can learn other options not covered in the manual.

Another very important issue when dealing with the file system is the *permissions* on your files and directories. Type `ls -al` in your home directory. You will see something like this:

```
drwxr-xr-x 18 yourname      1536 Jul 22 16:32 .
drwxr-xr-x 63 root          1536 Jul 17 09:16 ..
-rw-r--r--  1 yourname     4111 Jul 14 15:05 .cshrc
-rw-r--r--  1 yourname     1562 Nov 16 1994 .login
```

There might be more directories and files than these, but that is not important. Look at the 10 letters and minus signs in the left column. These letters give information about who has what sort of access to your files.

The leftmost letter tells us whether we are dealing with a file, a directory, or a link to a file (links to files will not be covered here). A `'d'` in the leftmost column tells us we are dealing with a directory; a `'-'` in the leftmost column indicates a file. The next 9 letters are either `'r'`, `'w'`, `'x'`, or `'-'`. The `'r'` indicates read permission, the `'w'` indicates write permission, and the `'x'` indicates execute permission. A `'-'` in a particular column indicates that the permission for that column is not set.

Columns two through four are your own permissions; columns five through seven are your group's permissions; and columns eight through ten are all user's permissions.

For instance, type `mkdir silly`. Remember that this creates a directory called `silly`. Now type `ls -al`. You should see something like this (again, you may have more files and directories than this):

```

drwxr-xr-x 18 yourname      1536 Jul 22 16:32 .
drwxr-xr-x 63 root         1536 Jul 17 09:16 ..
-rw-r--r--  1 yourname     4111 Jul 14 15:05 .cshrc
-rw-r--r--  1 yourname     1562 Nov 16 1994 .login
drwxr-xr-x  2 yourname      512 Jul 23 09:34 silly

```

Look at the permissions for the directory `silly`. The leftmost column contains a ‘d’ because `silly` is a directory. The ‘rwx’ in columns two through four indicate that you have read, write, and execute permissions on directory `silly`. The ‘r-x’ in columns five through seven and in columns eight through ten indicate that both your group and all users have read and execute access to `silly`, but neither has write access. (At first it might seem to be strange to have execute permission on a directory, but if you remove the execute permissions to a directory, you will not be able to access that directory.)

Now look at your permissions for your `.cshrc` file. There is a ‘-’ in the leftmost column because `.cshrc` is a file and not a directory. The ‘rw-’ in columns two through four indicate that you have read and write access to the file `.cshrc`, but you do not have execute access. (This makes sense since `.cshrc` is a text file and cannot be executed.) In addition, the ‘-r-’ in columns five through seven and columns eight through ten indicate that your group and all users have read access to your `.cshrc` file, but do not have permission to write or execute your `.cshrc` file. While allowing other people to read your `.cshrc` file is ok, you will want to change the permissions on other files, such as class assignments, so that only you have read, write, and execute permission.

To change permissions, you will use the Unix command `chmod nnn file`. This command will change the permissions of the specified file according to the numbers represented in `nnn`. (The first `n` represents the permissions for yourself, the second `n` represents the permissions for your group, and the third `n` represents the permissions for all users.

To determine which numbers you want to use for which `n`, think of each ‘rwx’ as being a number represented in binary. (‘r’ corresponds to the most significant bit and ‘x’ to the least significant bit.) For instance, ‘rwx’ corresponds to the binary number 111 which is the decimal number 7; ‘r-x’ corresponds to the binary number 101 which is the decimal number 5; and ‘r-’ corresponds to the binary number 100 which is the decimal number 4. So, you decided which permissions you want to set, determine the binary number which corresponds to those permissions (3 different binary numbers, one for you, one for your group, and one for all users), and then translate those three binary numbers into 3 decimal numbers. Those 3 decimal numbers are the three `n` which are required as arguments for `chmod`.

For example, let’s say you want to change the permissions on your `.cshrc` file so that no one can write to or execute it, and you can only read from it (but you can’t write to or execute it). Type `chmod 400 .cshrc` `<<Return>>` and then type `ls -al`. Your permissions for your `.cshrc` file should be changed so that the directory listing for that file looks like this:

```

-r-----  1 yourname     4111 Jul 14 15:05 .cshrc

```

You will probably want to write to your `.cshrc` file in the future, so you should change the permissions so that you have write access to it. To do that type `chmod 600 .cshrc`. Now, type `ls -al` to verify that the permissions have been changed. However, since there is no harm in allowing other users read access to your `.cshrc` file, you can change the permissions to permit that if you wish. So, if you want, change the permissions so that you have read and write access and all other users have read access by typing `chmod 644 .cshrc`. Just remember to always keep your class assignments read, write, and execute protected!

Finally, some naming conventions. Most operating systems have naming conventions for files that have a primary part and an extension, usually separated by a period. The primary part is typically the name that distinguishes the file, and the extension helps in classification. So, ‘x.c’ says that ‘x’ is a C program whereas ‘x.tex’ indicates a Tex or Latex file (we will encounter these later). Generally, startup files in your

home directory only have an extension such as ‘.emacs’ and ‘.login’. In most cases, following these naming conventions is a good idea.

## 2.3 More Unix and System Information

In addition to the Unix commands presented in previous sections, here are some other useful commands:

**grep** <string> <file> – searches through **file** for the specified **string**, and prints the lines of **file** where **string** is found.

**date** – displays the system date and time

**head** <file> – displays the first few lines of the specified **file**

**tail** <file> – displays the last few lines of the specified **file**

**diff** <file2> <file1> – displays line by line differences between **file1** and **file2**

**find** **pathname-list** **expression** – recursively descends the directory hierarchy for each **pathname** in the **pathname-list**, looking for files that match a logical **expression**

**quota -v** – displays your current disk quota as well as how much disk space you have left

For information on other useful Unix commands, see Appendix A.

Another advanced feature of Unix is multi-processing. Multi-processing means that multiple processes can be “running” on a single processor machine at the same time. Technically, the multiple processes are being switched on and off the cpu, but it appears that they are all running at the same time. This allows you to run jobs /emphin the background.

Ordinarily, when you run a program, you type the executable name, hit <<Return>>, and then wait while your program is executing. Once the program is done, the prompt appears again, and you can type in the next command. Oftentimes it is far more useful to run a job in the background. Running a job in the background means that your prompt will come back immediately, and you can run other jobs on the same machine. (You should note, however, that if you run more than one computationally intensive job on one machine at a time, expect your workstation to respond rather sluggishly.) Your job (the one running in the background) will execute normally, but you do not have to sit idle and wait for it. To run a job in the background, simply add an ‘&’ at the end of the command line. For instance, say we want to run a program named **a.out** in the background. Then we would type **a.out &**.

Running a job in the background is particularly helpful when you are using an editor such as **Emacs**. Let’s say you are writing a program using the **Emacs** editor (we’ll cover **Emacs** later). Since few of us are perfect programmers, chances are you will have to make changes to your program after the first time you try to compile it. When you start **Emacs**, the editor appears in a new Xwindow (not in your xterm), but does not automatically run in the background. Therefore, if you do not run **Emacs** in the background, you will have to edit your program, close **Emacs**, compile your program, reopen **Emacs**, edit some more, and so on. This can be tedious. On the other hand, if you run **Emacs** in the background, then you can switch from editing to compiling (and back again) by simply moving the mouse cursor from the **Emacs** window into your xterm to change the focus of the window. Because **Emacs** is running in the background, the prompt comes back and you can compile your program there without exiting **Emacs**.

Occasionally you will run a job in the background and that job will either “hang” or simply not execute properly and you will want to *kill* it. In order to kill an executing job, you need to know its process id (or pid). To find out the pid of a job you are running, type **ps** at the prompt. You will see a list of the jobs you

are currently running. To kill a job, identify the pid which corresponds to the job you wish to kill, and then type `kill <pid>` where `pid` is that pid for that job. Both `ps` and `kill` have other options, but they will not be covered here. If you want more information on those options, see their `man` pages.

Another useful Unix command is `tar`; `tar` archives and extracts multiple files onto a single “tarfile”. This is useful if you have large directory you wish to move or email to someone. To create a tarfile, type `tar -cvf <file> argdirectory`. This command takes the given `directory` and compacts it into the tarfile named `file`. The ‘c’ option here indicates that you are *creating* a tarfile. The ‘v’ option causes the name of each tarred file to be printed to the screen as it is tarred. The ‘f’ option indicates that you wish to have the tarred output written to the specified *file*.

To extract a tar file, type `tar -xvf file`. This command extracts the directory or files from the tarfile `file`. The ‘v’ and ‘f’ option mean the same as before. The ‘x’ option indicates that you are extracting the files.

# Chapter 3

## Default Files

If you have looked around in your directory by typing in `ls -a1F` you'll have found files starting with a "." These are the default files (or *user custom files*). The default files are used to set up your environment the way you want it. There are several such files in your home directory, but we will only discuss a few of them here. The files we will discuss here are `.login`, `.cshrc` and `.logout`.

`.login` is executed only once when you log into the computer.

`.cshrc` is executed each time you start a new shell as well as when you first log in.

`.logout` is executed only once when you logout of the computer.

These are not the only default files. Other default files include `.bbrc`, `.plan`, etc. As discussed before, one can tailor one's environment by putting appropriate code into the default files.

The next few pages will show you sample `.login`, `.cshrc` and `.twmrc` files. At first glance the code in these files can be baffling. A complete explanation of the code is beyond the scope of this user guide, but descriptions of some of the more useful commands have been included here.

### 3.1 The `.login` File

Shown below is a typical `.login` file.

```
# @(#)Login 1.1 86/07/09 SMI
#####
#
#      .login file
#
#      Read in after the .cshrc file when you log in.
#      Not read in for subsequent shells.  For setting up
#      terminal and global environment characteristics.
#
#####

#      terminal characteristics for remote terminals:
#
#      Leave lines for all but your remote terminal commented
```

```

#           out (or add a new line if your terminal does not appear).

if ($TERM != "sun") then
set noglob
eval 'tset -sQ -m dialup:?vt100 -m su:?vt100 -m dumb:?vt100 -m network:?vt100 $TERM'
unset noglob
endif

#           general terminal characteristics

#stty -crterase
#stty -tabs
#stty crt
stty erase '^h'
#stty werase '^?'
stty kill '^?'
#stty new

#           environment variables

setenv ENSCRIPIT '-fCourier10'
setenv MORE '-c'
setenv NEPRO "~/emacs_pro"
setenv PRINTER hp1
setenv TEXEDIT "/usr/local/bin/emacs -l/usr/lib/tek82/tex-start -estartline %d %s"
setenv TROFF "ptroff"
setenv TCAT "lpr"
setenv XENVIRONMENT $HOME/.Xdefaults

#           commands to perform at login

if ("tty" != "/dev/console") exit

if ( $term != xterm) echo "-----"
if ( $term != xterm) bb -n
if ( $term != xterm) echo "-----"
if ( $term != xterm) echo ""

#           commands to execute X windows by default
echo -n "X Windows? (^C to interrupt) "
sleep 5
echo "Starting X ..."
xinit /usr/bin/X11/xstart -- -zaphod
echo -n "Logout? (^C to interrupt) "
sleep 5
logout

```

## 3.2 The .cshrc File

Shown below is a .cshrc file. Your .cshrc file will probably initially have fewer aliases, user commands, set paths, etc.

```

## for Monet HOSTTYPE == iris4d# @(#)Cshrc 1.1 86/07/09 SMI
#####
#
#           .cshrc file

```

```

#
#       initial setups for both interactive and noninteractive
#       C-Shells
#
#####

#       set up search path

if ( $HOSTTYPE == sun4 || $HOSTTYPE == i386-linux ) then

    set MACHINE='uname -m'
    set RLAB_BIN = (/home/mom1/bynum/bin /home/mom1/bynum/bin/$MACHINE)
    set lpath = ( ) # add directories for local commands
    set path = (/usr/lang /usr/local /usr/ucb /usr/bin /bin /usr/local/bin )
    set path = ($path /usr/local/gnu/bin . ~/bin )
    set path = ($path $lpath /usr/bin/X11 )
    set path = ($path /usr/local/lib/sun4/ch_pr)
    set path = ($path /usr/openwin/bin )
    set path = ($path $RLAB_BIN)
    setenv VISUAL emacs
    setenv MAIL /usr/spool/mail/'whoami'
    setenv MAILDROP $MAIL

endif

if ( $HOSTTYPE == sun4 ) then
    setenv GCC_EXEC_PREFIX /usr/local/gnu/lib/gcc-lib/sparc-sun-sunos4.1.3/2.7.2/
    setenv ADA_INCLUDE_PATH /usr/local/gnu/adainclude
    setenv ADA_OBJECTS_PATH /usr/local/gnu/lib/gcc-lib/sparc-sun-sunos4.1.3/2.7.2/adalib
    setenv C_INCLUDE_PATH /usr/local/gnu/lib/gcc-lib/sparc-sun-sunos4.1.3/2.7.2/include
else if ( $HOSTTYPE == i386-linux ) then
    setenv GCC_EXEC_PREFIX /usr/local/gnu/lib/gcc-lib/i486-linux/2.7.2.1/
    setenv ADA_INCLUDE_PATH /usr/local/gnu/adainclude
    setenv ADA_OBJECTS_PATH /usr/local/gnu/lib/gcc-lib/i486-linux/2.7.2.1/adalib
    setenv C_INCLUDE_PATH /usr/local/gnu/lib/gcc-lib/i486-linux/2.7.2.1/include
endif

if ( $HOSTTYPE == iris4d ) then
    setenv VISUAL emacs
    set MACHINE='uname -m'
    set RLAB_BIN = (/home/mom1/bynum/bin /home/mom1/bynum/bin/$MACHINE)
    set lpath = ( ) # add directories for local commands
    set path=(/usr/local/bin/X11 /usr/bin/X11 /bin /sbin /usr/sbin)
    set path=(/usr/local/gnu/bin /usr/local/bin /usr/bsd . ~ $path)
    set path = ($path $RLAB_BIN)
    setenv VISUAL emacs
# set prompt="%m[%c]> "
endif

    setenv MANPATH /usr/man:/usr/local/man:/usr/local/gnu/man:/usr/local/interviews/man:/usr/lang/man:
    setenv MANPATH "$MANPATH":'/home/mom1/bynum/man'

#       cd path

set lcd = ( ) # add parents of frequently used directories
set cdpath = (.. ~ $lcd)

#       set this for all shells

set noclobber

#       set up environment variables

# set umask to 002 if other grads need write access

```

```

#umask 002
umask 022

#           skip remaining setup if not an interactive shell

if ($?USER == 0 || $?prompt == 0) exit

#           settings for interactive shells

set history=40
set savehist=40
set prompt="(\\!) %n%#m> "

#           other aliases

alias bye      logout
alias cd       'cd \\!*;echo $cwd'
alias cls      'clear'
alias copy     'cp -i'
alias h        history
alias !        history
alias cl       clear
alias em       'emacs \\!*&'
alias cp       'cp -i'
alias del      'rm -i'
alias dir      'ls -aF'
alias ls       'ls -al'
alias lsm      'ls -al | more'
alias psm      'ps -aux | more'
alias help     man
alias list     cat
alias lo       logout
alias locate   'find ~ -name \\!* -print'
alias m        more
alias mroe     more
alias rmoe     more
alias omre     more
alias moer     more
alias mv       'mv -i'
alias rm       'rm -i'
alias pwd      'echo $cwd'
alias type     more
alias up       'cd ..'

```

There are several useful commands in your `.cshrc` file which we will now cover. First, you will note that at the end of this `.cshrc` file are several lines which begin with the command `alias`. An alias such as `alias myalias '<some command>'` will alias `<some command>` to the word `myalias`. Now whenever you type `myalias`, `<some command>` is executed.

Aliases are useful if you prefer using the options with a Unix command, but you don't want to have to remember the options (and type them) each time you use it. For example, `alias copy 'cp -i'` in your `.cshrc` file will allow you to use `copy` to copy files instead of `cp`. The `-i` option prompts user for confirmation whenever the copy would overwrite an existing file. In addition, let's say you almost always mistype the word "more", spelling it "mroe" instead of "more". You can create an alias (there's one like this in the above file) so that when you type "mroe" the "more" command is executed. Aliases are also useful when you want to avoid typing long commands.

You may also want to change your xterm prompt. This is also done in your `.cshrc` file. Find the line in your `.cshrc` file that begins with `set prompt`. There are several ways you can change your prompt. For instance,

if you change the `set prompt` line to read `set prompt='\! %n%m> '` then the command line number is displayed (the `\!` does that), followed by your username the `@` symbol and the hostname of the machine you are using. For instance, say you have typed 146 command lines into your xterm, your username is smith, and you are on astro. Then your prompt would read `(147) smith@astro> .`

Another way to change your prompt is to modify the `set prompt` line to read `set prompt='%m[%c]>'`. This prompt will list your computer's hostname followed by the name of your current directory. For instance, if you are on astro in the directory garbage, this prompt will read `astro[garbage]>`. For more information on setting your prompts as well as other useful information about the `tcsh` shell, see the `man` page on `tcsh`. (In other words, at the prompt in your xterm type `man tcsh`.)

Because your `.cshrc` file is loaded only when you start a new shell or log in, any changes you make to the file will not be immediately visible. To see your changes, type `source .cshrc` at the prompt. This will reevaluate your `.cshrc` file. Note however, that some changes you make in your `.cshrc` file will not take effect until after you log off and log back on again.

**PATHNAMES:** One of the reasons you will sometimes edit your `.cshrc` file is to change your `PATH` variable. You can see what your current `PATH` variable and all other variables are set to by typing `setenv` by itself. Whenever you are unable to run an application because it can't be found, it's usually because you need to add something to the `PATH` variable. The `PATH` is simply a list of paths that the system will search whenever you type a command. For example, if you type `javac` for the Java compiler, you might not get anything if the path to this compiler is not in your collection of paths in the `PATH` variable. Whenever newly-installed software is announced, the appropriate path is usually also announced (on the bulletin board).

## Chapter 4

# Editing: the Emacs editor

The commonly used Unix editors are `vi` and `Emacs`, of which `Emacs` is the most powerful. If you are already familiar with `Emacs`, all you need to know is that we use GNU-`emacs` from the Free Software Foundation (a little different from VMS `Emacs`) and you can skip this chapter.

### 4.1 Getting Started

Start by creating a sample file to work with: `cp .cshrc junk`. You should have a `.emacs` file already in your directory; if not, type `cp /usr/local/emacs/etc/.emacs .emacs` to copy a special startup file. Your `.login` and `.cshrc` files should be already setup for `Emacs`.

You bring up `Emacs` by typing `emacs junk &`. (Again, remember that the `&` runs `Emacs` in the background.) This will bring up a new X window. At the top border of this window is the label `GNUemacs`. Below that is a menu bar which looks like this:

```
Buffers Files Tools Edit Search Help
```

At the bottom of the window is a *menu buffer* which looks something like this

```
-----Emacs: junk(Fundamental)--L1--Top-----  
For information about the GNU Project and its goals, type C-h C-p.
```

The rest of the window contains your file, `junk`.

The cursor should be at the very beginning of the file. You can move the cursor in a variety of ways. You can use the arrow keys on your keyboard to move the cursor up, down, right, or left. In addition, if you move the mouse cursor and then click someplace in the file, the cursor will move there. Finally, you can use the following commands:

- «ctrl-f» – move forward a character
- «ctrl-b» – move backward a character
- «ctrl-n» – move to next line
- «ctrl-p» – move to previous line
- «ctrl-a» – move to beginning of line
- «ctrl-e» – move to end of line

## 4.2 Saving and Exiting

Next, enter some text of your own anywhere in the file (say at the top of the file) by simply typing in something. Now save the file by typing `<<ctrl-x>>` and `<<ctrl-s>>` one after the other (we will write this as `<<ctrl-x>><<ctrl-s>>`). You can also save your file by clicking on **Files** on the menu bar at the top of the window. A pull-down menu will appear. Now click on the option **Save Buffer**. For either method, in the menu buffer at the bottom of the window will appear the words **Saving file /home/mom2/your-username/junk** quickly followed by the words **Wrote /home/mom2/your-username/junk**.

You quit **Emacs** by typing `<<ctrl-x>><<ctrl-c>>`. In addition, you can also exit **Emacs** by opening the **Files** menu and selecting **Exit Emacs**. When you have exited **Emacs**, the X window containing **Emacs** will disappear.

**Emacs** has an on-line tutorial. To get into the tutorial (you have to be “in” **Emacs**) type `<<ctrl-h>>` `t` and follow the instructions that you’ll see. When you have learned what you want (or when you get bored), type `<<ctrl-x>><<ctrl-s>>` followed by a `<<ctrl-x>><<ctrl-c>>` to quit. In addition, a list of useful **Emacs** commands may be found in Appendix B.

## 4.3 Buffers

One of the powerful aspects of **Emacs** is that it allows you to have multiple *buffers*. In other words, you can open and edit different files (within different **Emacs** buffers) without every exiting and restarting **Emacs**. If you are no longer in **Emacs**, then start it up again by typing `emacs junk &`. Again, your file `junk` will be displayed.

Now save your `junk` file under a different name. This can be done in 2 ways. You could either go to the **Files** menu and select the option **Save Buffer as...** or you can type `<<ctrl-x>><<ctrl-w>>`. Either way, at the bottom of your **Emacs** window, in the menu buffer will appear

```
Write file: ~/
```

Type `junk2` and press `<<Return>>`. You now have 2 files, `junk` and `junk2`. The file which is currently visible in your window is `junk2`. To reopen the file `junk`, type `<<ctrl-x>><<ctrl-f>>`. In the menu buffer, you will see the words

```
Find file: ~
```

Type `junk` and press `<<Return>>`. Again you will see only the file `junk`, but the buffer for `junk2` is *still* open in **Emacs**. To see `junk2`, (to switch between buffers), select **Buffers** from the menu bar. You will see a menu whose options are your buffers (`junk` or `junk2`). Select the buffer you wish to view (in this case `junk2`). Alternatively, to see the **Buffer** menu, hold down the `<<ctrl>>` key and then click and hold the left mouse button. Then **Buffer Menu** will appear in your window and you can select which buffer you wish to edit.

## 4.4 Deleting and Undoing

Both `<<Del>>` and `<<Bksp>>` will delete characters behind the cursor one at a time. `<<ctrl-d>>` will delete characters in front of the cursor one at a time. `<<ctrl-k>>` will delete to end of line, so be careful with this one!

Suppose you typed `<<ctrl-k>>` and deleted the end of a line accidentally. Now you wish to *undo* that mistake. `<<ctrl-_->>` will undo the change or any other unintended changes you might have made inadvertently! If you have a hard time remembering all these `<<ctrl>>` characters, remember you can use the menu bar. To undo using the menu bar, select **Edit** and then **Undo**. **Emacs** allows you to undo more than just your most recent change; in fact, you can undo quite a lot using `<<ctrl-_->>`.

Another *very* useful feature to know is the `<<ctrl-g>>` command. Use it whenever

- You are in the middle of issuing a Emacs command you don't want.
- When Emacs is hung, i.e. if Emacs gets into an infinite (or simply very long) computation which you don't want to finish.
- You have typed an <<Esc>> by mistake.

In short <<ctrl-g>> cancels whatever Emacs command you were issuing.

## 4.5 Moving Blocks

Appendix B lists the cursor movement commands that move the cursor faster than a character at a time (e.g., <<ctrl-v>> and <<Page Down>> moves a screenful forward, <<ESC-v>> and <<Page Up>> move a screenful backward, <<Home>> moves you to the beginning of the buffer, and <<End>> moves you to the end of your buffer). If you have not used an editor which deletes a whole region of text by first defining it and then moving it to a separate buffer, then the following will introduce you to the concepts.

Start by creating a new file using Emacs. If you are already in Emacs, type <<ctrl-x>><<ctrl-f>> and then type the file name junk3 and hit <<Return>>. If you are not in Emacs, type `emacs junk3 &` at the prompt. Now you have an open Emacs window with a blank (new) file named junk3.

Next, enter the following text:

```
asdf2345qwer
i never knew i could have so much fun with Unix
*Yourname
```

There are basically 2 ways in which you can cut, copy, and paste blocks of text in Emacs. One way is by strictly using <<ctrl>> characters and the other is using the mouse in conjunction with the <<ctrl>> characters. First we will go over the strictly <<ctrl>> character method. Position the cursor over the number '3' in the first line and type <<ctrl-@>>. By doing this, a 'mark' will be set just before the number '3'. Next, move the cursor over the letter 'w' in the word 'knew'. (Some of the text may be highlighted.) Now type <<ctrl-w>>. You get

```
asdf2w i could have so much fun with Unix
*Yourname
```

All the characters between the mark and the cursor have been cut to a buffer (the killbuffer or yankbuffer, as it is called). The <<ctrl-w>> command always removes the region defined by these boundaries to the killbuffer (one has to be very careful with <<ctrl-w>>!). Now position the cursor over the character '\*' and type <<ctrl-y>>. You get

```
asdf2w i could have so much fun with Unix
345qwer
i never kne*Yourname
```

So, <<ctrl-y>> pulls text from the killbuffer and pastes it just before the current position of the cursor.

Before we learn the "mouse" method of doing this, hold down the <<ctrl>> key and hit the <<\_>> (the underscore) twice. This will undo the paste you have just done as well as the cut. Again, your text should look like

```
asdf2345qwer
i never knew i could have so much fun with Unix
*Yourname
```

Now move the mouse cursor to the number 2. Click and hold the left mouse button. Drag the mouse cursor to the space after the word “knew”. Release the mouse. The text to be cut or copied should be highlighted.

At this point you can do 2 things to copy or cut and then paste the highlighted text. You can click on the menu bar under **Edit** and then select the appropriate option or use the `<<ctrl>>` characters to copy, cut and paste. Since the menu option is fairly straightforward, only the other option will be covered here.

Move the mouse cursor to after the word “Unix” and click here to move the cursor to here. (Your text will no longer be highlighted.) Now type `<<ctrl-y>>`. The previously highlighted text has been copied (but not cut) and pasted to here. Your file should now read

```
asdf2345qwer
i never knew i could have so much fun with Unix2345qwer
i never knew
*Yourname
```

Note that the original text *has not been deleted*. Let’s say that you now realize that you do not like the way this file looks and you want to delete the most recently pasted text. You can do simply do an undo, or you can highlight the text again (using `<<ctrl-@>>` or the mouse) and then delete it by typing `<<ctrl-w>>`.

## 4.6 More on Emacs

Between the tutorial and Appendix B you should have enough to satisfy most of your editing needs. A few not-so-obvious things may be mentioned here.

Remember that any command, once started, can be immediately canceled using `<<ctrl-g>>`. The usual Unix control characters `<<ctrl-d>>` and `<<ctrl-c>>` have different meanings in **Emacs**. Several esoteric commands are available at a line-editing level by hitting `<<ESC>>` `x` followed by the full name of the command. For example, if you have defined a region using `<<ctrl-@>>` and the cursor, you can convert every letter in the region to upper case by typing `<<ESC>>` `x` followed by `upcase-region`. There are many such powerful commands – these are listed in the emacs manual.

To search for a string in your document type `<<ctrl-s>>`. The string “I-search” appears. As you type in the string you are looking for, **Emacs** tries to match it with a string in the document. Type `<<ctrl-s>>` to repeat the search. `<<ctrl-s>>` searches for the string *after* the current cursor position. Type `<<ctrl-s>>` to find the next occurrence of the string. `<<ctrl-r>>` searches in the reverse direction. `<<Esc>>` terminates a search (`<<ctrl-g>>` will work too).

Finally, you can place special key-binding commands in the ‘.emacs’ file to enable you to redefine keys and to allow you to use function keys and keypad keys on your keyboard. For a complete list of functions, consult the Emacs manual.

## Chapter 5

# Creating Your Own Web Page

To create your own webpage, first make a directory (in your home directory) named `public.html`. Within that subdirectory, you must create an HTML file named `index.html`. This file (`index`) will be your home page.

Rather than creating a home page from scratch, you may find it useful to copy someone else's homepage and modify it for yourself. Start up Netscape by typing `netscape &` at the prompt. Now use the internet to go to the home page you wish to copy and make into your own. Once you have found that page, select File from the menu bar at the top of the Netscape window and Save As... from the pull-down menu. A dialog box will appear. In this box, under the label "Selection", type in the name of your `index.html` file with the correct directory path: `/home/mom2/your_userid/public.html/index.html` and then click on the "ok" button.

The file "index.html" is now in your `public.html` directory. You can now modify it to make it your own web page. To learn more about how to modify your html to get it to look the way you want, click "Help" on the menu bar at the top of your Netscape window and then select "How to Create Web Services" from the pull-down menu. The web page which comes up provides links to many sites from which you can learn how to create your web page. Alternatively, visit this site: <http://home.netscape.com/home/how-to-create-web-services.html>

Once you have created your "index.html" file in the `public.html` directory, then to see your web page, go back into Netscape and go to the site: [http://www.cs.wm.edu/~your\\_userid](http://www.cs.wm.edu/~your_userid) When you visit this site, your home directory is automatically searched for the "index.html" file within the `public.html` directory when it sees the `~your_userid`.

# Chapter 6

## L<sup>A</sup>T<sub>E</sub>X

L<sup>A</sup>T<sub>E</sub>X is a word processing program that is specially suited to scientific writing. Virtually everyone in Computer Science uses it (or its predecessor, T<sub>E</sub>X) nowadays; L<sup>A</sup>T<sub>E</sub>X is fast becoming the standard for the electronic submission of papers. The new computer science student is well-advised to start learning this document preparation system as early as possible (all professors prefer homework done in L<sup>A</sup>T<sub>E</sub>X; some require it). This primer has been written entirely in L<sup>A</sup>T<sub>E</sub>X.

The T<sub>E</sub>X in L<sup>A</sup>T<sub>E</sub>X is actually an uppercase form of the greek  $\tau\epsilon\chi$ , and since the greek  $\chi$  is pronounced as chi, T<sub>E</sub>X gets the pronunciation ‘tecch’ or ‘teck’, making *lah-teck*, *lah-teck*, and *lay-teck* the logical choices for L<sup>A</sup>T<sub>E</sub>X.

The T<sub>E</sub>X in L<sup>A</sup>T<sub>E</sub>X refers to Donald Knuth’s T<sub>E</sub>X typesetting system. T<sub>E</sub>X is a sophisticated program designed to produce high-quality typesetting, especially for mathematical text. The L<sup>A</sup>T<sub>E</sub>X software is comprised of a collection of high level macros of T<sub>E</sub>X, and is one of the many document preparation systems built around T<sub>E</sub>X. L<sup>A</sup>T<sub>E</sub>X adds to T<sub>E</sub>X a collection of commands that simplify typesetting by letting the user concentrate on the structure of the text rather than on formatting commands. Sooner or later you will be writing a paper, replete with symbols, tables and references. Using L<sup>A</sup>T<sub>E</sub>X you can produce a quality document (such as this one - ahem!) with relative ease. It’s a little painful getting started, but hang in there and it will soon become second nature to you.

Note: Although this primer will help you get started with L<sup>A</sup>T<sub>E</sub>X, you should probably invest in the L<sup>A</sup>T<sub>E</sub>X manual, available at the College Bookstore, at *Waldenbooks* in Patrick Henry Mall, or at *Barnes and Noble* next to Patrick Henry Mall. In addition, the department has invested in several L<sup>A</sup>T<sub>E</sub>X manuals which are in both rooms 111 and 121. These manuals are there for you to use, but please do not remove them from these rooms!

### 6.1 Getting Started With L<sup>A</sup>T<sub>E</sub>X

To get started in L<sup>A</sup>T<sub>E</sub>X, open a new file in Emacs by typing `emacs temp.tex &`. Your Emacs window should open with an empty (new) file in it named `temp.tex`.

The following lines may look a little strange at first, but they are both the information for your paper as well as L<sup>A</sup>T<sub>E</sub>X commands which initialize and format your paper. Write these lines *exactly as they appear here* into your opened file.

```
\documentclass[11pt]{article}
\usepackage{fullpage}
```

```

\pagestyle{empty}
\parindent 0.0in
\begin{document}

\begin{center}
{\LARGE \textbf{Welcome to \LaTeX{}}!}
\end{center}

```

Start by just typing some simple text, such as this\dots

Now type something in `\emph{italic}`.

Finally, leave the above 3 lines blank and four extra spaces here just to see what Latex does with it.

`%` this is a comment. Latex ignores all lines beginning with `%`.

See, `\LaTeX{}` isn't so bad!!

```

\end{document}

```

Now save this file. You are now ready to see how  $\LaTeX$  processes this file and produces its output. To invoke the  $\LaTeX$  program, type `latex temp`. This command is analogous to a compilation of a C program. When invoked,  $\LaTeX$  will check your file for errors (not spelling or grammar errors, but  $\LaTeX$  errors), and then produce a file called `temp.dvi`. When you type `latex temp`, you should see output which looks something like this:

```

This is TeX, Version 3.14159 (C version 6.1)
(temp.tex
LaTeX2e <1996/06/01>
Hyphenation patterns for english, german, loaded.
(/home/mom0/teTeX/texmf/tex/latex/base/article.cls
Document Class: article 1996/05/26 v1.3r Standard LaTeX document class
(/home/mom0/teTeX/texmf/tex/latex/base/size11.clo))
(/home/mom0/teTeX/texmf/tex/latex/misc/fullpage.sty
Style Option FULLPAGE Version 2 as of 15 Dec 1988
)
No file temp.aux.
[1] (temp.aux) )
Output written on temp.dvi (1 page, 744 bytes).
Transcript written on temp.log.

```

If you receive an error, you will get an error message (which might be pretty vague) followed by a  $\LaTeX$  prompt. At the `?`, type `x`  $\llcorner$ Return $\lrcorner$  and then carefully check your Emacs file to make sure you copied the above text *exactly*.

Once your  $\LaTeX$  file has no errors, type `ls` to check which files were created by  $\LaTeX$ . In your directory, there should be your original `temp.tex` as well as the `temp.dvi` file created by  $\LaTeX$ , and possibly `temp.aux` and `temp.log`. The suffix `.dvi` stands for *device independent* and is a standardized encoding of your file, as processed by  $\LaTeX$ , for different printers.

Next, type `dvips temp`. The command `dvips temp` converts your  $\LaTeX$  `temp.dvi` file into a PostScript file. By typing `dvips temp` you have just sent that file to your default printer. You can check this by typing `lpq`,

which should show your job printing or in the queue (or that the printer is down!). In any case, assuming your print job was first in line, it should not take long to print. Once your file is printed, it should look like this:

## Welcome to L<sup>A</sup>T<sub>E</sub>X!

Start by just typing some simple text, such as this. . .

Now type something in *italic*.

Finally, leave the above 3 lines blank and four extra spaces here just to see what Latex does with it. See, L<sup>A</sup>T<sub>E</sub>X isn't so bad!!

You are now half-way along to mastering L<sup>A</sup>T<sub>E</sub>X!

There are many characteristics of L<sup>A</sup>T<sub>E</sub>X input files which you should understand. First and foremost, there are certain reserved characters which are defined by T<sub>E</sub>X to perform some special functions, and if used in incorrectly can cause some surprising results.

The *legal* (non-reserved) characters are the upper- and lowercase letters, the ten digits 0...9 and the following 16 punctuation characters:

. : ; , ? ! ' ' ( ) [ ] - / \* @

The ten special or *reserved* characters are

# \$ % & ~ \_ ^ \ { }

The reserved characters are used in L<sup>A</sup>T<sub>E</sub>X commands, but there are ways for you to print out these characters should you need to. The special meanings of these reserved characters will be explained throughout this chapter on L<sup>A</sup>T<sub>E</sub>X.

One reserved character which you used in your first L<sup>A</sup>T<sub>E</sub>X file is the '%' character. The '%' character is used for making comments in an input file, and so any text following the '%' character *on the same line* will not appear in the output. This became apparent when you printed out your L<sup>A</sup>T<sub>E</sub>X file. Similarly, any character string following a *backslash* or '\' character is interpreted as a control sequence by L<sup>A</sup>T<sub>E</sub>X. Arguments of these commands are enclosed in curly brackets. For instance, in your file, you used the L<sup>A</sup>T<sub>E</sub>X command "\emph" to print the word "italic" in italic. You also used the "\LaTeX{ }" to print out the word LaTeX so that it looks like L<sup>A</sup>T<sub>E</sub>X.

The second feature to be pointed out is the structure of the input file. A L<sup>A</sup>T<sub>E</sub>X input file basically has two parts – the text of the document and the *preamble*. The preamble contains the set of declarations and commands which affect the style and format of the whole document. For instance, a command specifying the document style, say, to be either single column or double column would be given in the preamble. L<sup>A</sup>T<sub>E</sub>X recognizes the end of the preamble and the beginning of the document when it encounters the command \begin{document}. Likewise, the end of the document is marked by the \end{document} command. Any higher level macros to be loaded, or new user specified commands are all declared in the preamble.

The third characteristic, as you might have noticed, is the way L<sup>A</sup>T<sub>E</sub>X *ignores unnecessary spaces between words* in the document. This could be observed in the temp.tex file. Whether you leave one space or five between words, during the process of typesetting, L<sup>A</sup>T<sub>E</sub>X ignores all spaces except for the required single space. Also, as temp.tex shows, L<sup>A</sup>T<sub>E</sub>X *interprets a complete blank line or more between text as an indication for paragraph breaking*. Again, one line or twenty blank lines generates the same vertical space between paragraphs. You should note that because there were no entirely blank lines between the commented line (the one beginning with a %) and the next line (that one that begins with "See"), that text line (the one beginning with "See") was put at the end of the line before the comment. LaTeX simply assumed that the commented line came in the middle of a paragraph.

## 6.2 A More Complicated Example

We will now look at a more complex input file to L<sup>A</sup>T<sub>E</sub>X and explain some of the most useful features of L<sup>A</sup>T<sub>E</sub>X. As an exercise, you might like to type in the following input file (as a `something.tex` file) and L<sup>A</sup>T<sub>E</sub>X it.

```
\documentclass[11pt]{article}
% set page boundaries and other stuff
\topmargin      0.25truein
\oddsidemargin  0.0truein
\evensidemargin 0.0truein
\textheight     8.5truein
\textwidth      6.5truein
\footskip        0.6truein
\headheight     0.0truein
\headsep        0.0truein
\parskip 4pt plus 1pt
\parindent=0pt
```

```
% document starts here
\begin{document}
```

```
% a nice title to start off with
\begin{center}
{\large\bf A sample input file}\\
{\it \today}
\end{center}
```

```
% Lines starting with '%' character are construed as comment statements.
The text presented in this input file is meant to give you an
understanding of how ordinary text is input in the file. Comparing
this input file, to the output it generates,
%(which follows this file)
will give you a better understanding of how to produce your own file.
```

Notice how a blank line skipped in the input file caused a paragraph break in the output, while that long blank space in the last paragraph caused no change in the output file. Also noticeable is the fact that the indentation given at the beginning of this paragraph, made no difference to the output file, since that is controlled by the specification for indentation given in the preamble\footnote{This will be explained in detail later.Notice how a footnote is specified in the input file}.

Reserved characters like `\%`, `\&`, `\#`, `\$`, `\{` and `\}` are produced by preceding them with the control character `\verb" \ "`. You can `{\it interchange}` `{\bf fonts}` `{\sf and}` `{\tiny sizes}` in this way. `{\em The default type style is {\rm roman} and the default type size is {\rm ten-point}}`.

```
% First bit of fancy stuff
```



- This is the first item of an itemized list. Notice the bullet mark denoting it.
  1. You can also enumerate items.
  2. This environment numbers the items.
- This is the second item of the original list.

Just when you thought you had seen the last of calculus:

$$\lim_{\alpha \rightarrow \infty} \frac{1}{\alpha} \int_0^\alpha \log(\exp[\alpha])^{-1} d\alpha = -1 \tag{6.1}$$

four spaces left there.

two lines and six spaces.

The document is finally closed by the `\end{document}` command.

In the following discussion we will go through this input file and explain some of the important commands.

## 6.3 Understanding the Input File

To begin with, the various statements in the preamble will be discussed in detail. Some of these statements are mandatory, while others are optional and if omitted may take default values. As explained previously, the declarations and commands which affect the whole document are specified in the preamble. Following this, most of the commands and *environments* which are often used in text documents will be explained.

### The Preamble

The text of every document starts with a `\begin{document}` command ends with an `\end{document}` command. Everything following the `\end{document}` command is ignored. The part of the input file that precedes the `\begin{document}` is called the *preamble*.

The preamble is discussed in greater detail in APPENDIX C.

### 6.3.1 Common Commands

As mentioned before, any character string following the control sequence `\` is interpreted by  $\LaTeX$  as a command or declaration.

#### Fonts

A *font* is a particular size and style from a *font family*. There are two families of fonts which  $\LaTeX$  supports. They are the CM or Computer Modern font family and AM or Almost Computer Modern font family. The laser printer supports both the AM and CM fonts. If you use any of the predefined type styles or sizes of  $\LaTeX$  explained below, you will be accessing CM fonts. To use any of the AM fonts, the font would have to be explicitly defined in the preamble. There are seven basic type styles supported by  $\LaTeX$ . The default type style is called “roman”, the type style used for this sentence. A different type style could be selected at any point in the document by giving the declaration pertaining to that particular style and enclosing the extent of the text in curly braces. The six other type styles are shown below, with the declarations that selects them.

<code>{\bf This is bold type style.}</code>	<b>This is bold type style.</b>
<code>{\sf This is sans serif type style.}</code>	This is sans serif type style.
<code>{\sl This is slanted type style.}</code>	<i>This is slanted type style.</i>
<code>{\it This is italic type style.}</code>	<i>This is italic type style.</i>
<code>{\sc This is Small Caps type style.}</code>	THIS IS SMALL CAPS TYPE STYLE.
<code>{\tt This is typewriter type style.}</code>	This is typewriter type style.

The default roman type style is selected with the declaration `\rm`. The *italics* type style used for emphasizing text could be invoked by the declaration `\em` also. The  $\LaTeX$  declarations for changing the type size are illustrated below:

<code>\tiny</code>	Size	<code>\normalsize</code>	Size	<code>\LARGE</code>	Size
<code>\scriptsize</code>	Size	<code>\large</code>	Size	<code>\huge</code>	Size
<code>\footnotesize</code>	Size	<code>\Large</code>	Size	<code>\Huge</code>	Size
<code>\small</code>	Size				Size

These size-changing declarations specify the roman style, regardless of the style currently in effect. For `\large` bold letters, you must type `\large\bf`, *not* `\bf\large`.

#### Spaces



`\noindent`

Suppresses the paragraph indentation when used at the beginning of the paragraph.

`\newpage`

When one-column pages are being produced, this command ends the current paragraph and the current page. When two-column text is being produced, `\newpage` ends the current column rather than the current page; `\clearpage` ends the page, producing a blank right-hand column if necessary.

`\twocolumn[text]`

Starts a newpage by executing `\clearpage` and begins typesetting in two-column format. If the *text* argument is present, it is typeset in a double-column-wide parbox at the top of the new page.

`\onecolumn`

Starts a new page by executing `\clearpage` and begins typesetting in single-column format.

### 6.3.2 Environments

An *environment* is a  $\LaTeX$  construct used for displaying material, such as quotes, verses, lists, equations and the like. An environment has the form:

```
\begin{name} ... \end{name}
```

where *name* denotes the name of the environment.

#### Quotations

$\LaTeX$  provides two different environments for displaying quotations. The `quote` environment is used for a short quotation, while the `quotation` environment is used for quotations of more than one paragraph. The two environments are illustrated below:

The `quote` environment is as shown below:

```
\begin{quote}
This is a quote. {\em Shaji John}
\end{quote}
```

The `quotation` environment differs as shown:

```
\begin{quotation}
This happens to be a quotation even though
... the literal sense.
```

But this ... the `quotation` environment.

```
\end{quotation}
```

#### Lists

There are three list-making environments in  $\LaTeX$ : `itemize`, `enumerate`, and `description`. In all three, each new list item is begun with an `\item` command. The `itemize` and `enumerate` environments are used as illustrated in the sample file on pages 10 and 11. In the `description` environment, you specify the item labels with an optional argument to the `\item` command, enclosed in square brackets. The use of this environment is illustrated below:

The `quote` environment is as shown below:

This is a quote. *Shaji John*

The `quotation` environment differs as shown:

This happens to be a quotation even though you may differ with me in the literal sense.

But this is how you use the `quotation` environment.

Here is how you use this environment:  
`\begin{description}`  
`\item[pain] Trying to ... this primer.`  
`\item[primer] The painful ... manual.`  
`\end{description}`

Here is how you use this environment:  
**pain** Trying to read and understand this primer.  
**primer** The painful alternative to a reference manual.

## 6.4 Math Mode

L<sup>A</sup>T<sub>E</sub>X main sophistication lies in its ability to typeset mathematical text. This section of the primer is concerned with the same, to introduce the various environments, structures and symbols which are available in this mode, which makes this typesetting software non pariel. Before getting to them, let me digress to discuss the various *modes* in which L<sup>A</sup>T<sub>E</sub>X operates.

As L<sup>A</sup>T<sub>E</sub>X processes your input text, it is always in one of three modes: paragraph mode, math mode, or left-to-right (LR) mode. Most of what we encountered till now was processed in paragraph mode, where L<sup>A</sup>T<sub>E</sub>X regards your input as a sequence of words and sentences to be broken into lines, paragraphs and pages. But in order to generate a mathematical formula or represent an equation, L<sup>A</sup>T<sub>E</sub>X has to be in math mode. When in math mode, it regards letters in the input file to be mathematical symbols, treating “is” as the product of *i* and *s*, and ignores any space characters between them. LR mode is similar to paragraph mode in the sense that it considers your input to be a string of words with spaces between them. But the difference is that L<sup>A</sup>T<sub>E</sub>X never starts a new line in LR mode; the output keeps going from left to right. If you wish to learn more about LR mode, check one of the manuals in the lab.

### 6.4.1 Math Environments

There are three environments which put L<sup>A</sup>T<sub>E</sub>X in math mode – viz., `math`, `displaymath` and `equation`. The `math` environment is used when you have to use formulas in running text, called an *in-text* formula. In addition to the `\begin ... \end` construction, there are two shorter forms of invoking the `math` environment, namely `\(...\)` or `$. . . $`, which are more appropriate when the in-text formula is rather short. The other two environments are used when you have to *display* the formula or equation, the difference between the two being a counter which is associated with the `equation` environment, causing the displayed equation to be numbered. The `displaymath` environment, which has the short form `\[. . .\]`, produces an displayed formula which is not numbered.

The formula  $(a+b)^2 = a^2 + 2ab + b^2$  could be displayed in two ways.  
`\[ (a+b)^2 = a^2 + 2ab + b^2 \]`  
or as a numbered equation as:  
`\begin{equation}`  
`(a+b)^2 = a^2 + 2ab + b^2`  
`\end{equation}`

The formula  $(a + b)^2 = a^2 + 2ab + b^2$  could be displayed in two ways.

$$(a + b)^2 = a^2 + 2ab + b^2$$

or as a numbered equation as:

$$(a + b)^2 = a^2 + 2ab + b^2 \quad (6.2)$$

As mentioned earlier, L<sup>A</sup>T<sub>E</sub>X ignores spaces in the input when it is in math mode. But there are times when it is necessary to add a little space (or even at times remove some space) to make a formula look just right. The following four commands add the amount of horizontal space shown between the vertical lines:

<code>\,</code>	thin space	<code>\:</code>	medium space
<code>\!</code>	negative thin space	<code>\;</code>	thick space

The `\!` acts like a backspace, removing the same amount of space that `\,` adds. While the `\,` command can be used in any mode, the others can appear only in math mode.

## 6.4.2 Common Structures

### Subscripts and Superscripts

Subscripts and superscripts are made with the `_` and `^` commands.

`$x^y$`  is the same as  `$x_{1}^y$`  is the same as  `$x_1^y$`

### Fractions

Fractions denoted by the `/` symbol are made in the obvious way. But in the case of large fractions especially in displayed formulas, the `\frac` command is used. This has two arguments — the numerator and the denominator.

`$x = \frac{y+z/2}{1 + \frac{y}{z+1}}$`

$$x = \frac{y + z/2}{1 + \frac{y}{z+1}}$$

### Summations and Integrals

Summations and integrals, invoked by `\sum` and `\int` respectively, which involve subscript-sized expressions that appear above and below them are typed as ordinary subscripts and superscripts.

The in-text formula

`$\sum_{i=1}^n x_i = \int_0^1 f$`

appears different when displayed:

`$\sum_{i=1}^n x_i = \int_0^1 f$`

The in-text formula  $\sum_{i=1}^n x_i = \int_0^1 f$  appears different when displayed:

$$\sum_{i=1}^n x_i = \int_0^1 f$$

### Roots

The `\sqrt` command produces the square root of its argument. It has an optional first argument for producing other roots.

`$\sqrt{x+y}$`  ..  `$\sqrt[n]{2}$`

A square root  $\sqrt{x+y}$  and an  $n$ th root  $\sqrt[n]{2}$ .

### Ellipsis

An ellipsis is a sequence of three dots used, in ordinary text to denote omitted material. This is produced by the `\ldots` command, and works in any mode. For math mode,  $\LaTeX$  provides three more ellipsis, which find their use in arrays etc. They are `\cdots`, `\vdots`, and `\ddots` for centered, vertical, and diagonal ellipsis respectively.

... low ellipsis:  `$x_1, \ldots, x_n$`

... centered ellipsis:  `$a + \cdots + z$` .

Compare the use of a low ellipsis:  $x_1, \dots, x_n$  with that of a centered ellipsis:  $a + \cdots + z$ .

### Log-like functions

There are certain functions like “log”, “lim”, etc., which when used in math mode *should* be set in roman type then as the product of three quantities  $l$ ,  $o$ , and  $g$ , printed as “log”. To take care of this aspect,  $\LaTeX$  provides corresponding commands for all these functions (shown below).

<code>\arccos</code>	<code>\cos</code>	<code>\csc</code>	<code>\exp</code>	<code>\ker</code>	<code>\limsup</code>	<code>\min</code>	<code>\sinh</code>
<code>\arcsin</code>	<code>\cosh</code>	<code>\deg</code>	<code>\gcd</code>	<code>\lg</code>	<code>\ln</code>	<code>\Pr</code>	<code>\sup</code>
<code>\arctan</code>	<code>\cot</code>	<code>\det</code>	<code>\hom</code>	<code>\lim</code>	<code>\log</code>	<code>\sec</code>	<code>\tan</code>
<code>\arg</code>	<code>\coth</code>	<code>\dim</code>	<code>\inf</code>	<code>\liminf</code>	<code>\max</code>	<code>\sin</code>	<code>\tanh</code>

An example of the log function is:  
`\[ \log xy = \log x + \log y \]`

An example of the log function is:  
 $\log xy = \log x + \log y$

## Over- and Underlining

The `\overline` command puts a horizontal line above its argument, while the `\underline` command underlines the argument. The `\underline` command works in all three modes, while `\overline` works only in math mode<sup>3</sup>.

`\underline{This}` is an example of  
`\( \overline{x^2 + 1} \)`.

This is an example of  $\overline{x^2 + 1}$ .

Analogous to the above commands are the `\overbrace` and `\underbrace` commands which put horizontal braces above or below an expression.

### 6.4.3 Arrays

Arrays are produced with the `array` environment. It has a single argument that specifies the number of columns and the alignment of items within the columns. For each column in the array, there is a single letter in the argument that specifies how items in the column should be positioned: `c` for centered, `l` for flush left, or `r` for flush right. Within the body of the environment, adjacent rows are separated by a `\\` command and adjacent items within a row are separated by an `&` character.

```
\[ \begin{array}{llcl}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array} \]
```

The `array` environment can be used only in math mode and is meant for arrays of formulas. The next section describes an analogous `tabular` environment for making arrays of ordinary text items.

## Matrices and Determinants

Arrays like matrices and determinants are produced by putting large parentheses or vertical lines (called as *delimiters*) around an array. To make these delimiters, big enough to enclose the array it is trying to delimit, two commands `\left` or `\right` are used before the delimiter. The `\left` and `\right` commands should come in matching pairs, but the matching delimiters need not be the same. Shown below is the same array illustrated above, but as a matrix with the commands for delimiters added.

```
\[ x = \left( \begin{array}{llcl}
a_{11} & a_{12} & \ldots & a_{1n} \\
a_{21} & a_{22} & \ldots & a_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m1} & a_{m2} & \ldots & a_{mn}
\end{array} \right)
```

<sup>3</sup>In the case of putting a bar over an  $x$  to form  $\bar{x}$ , one should use `\bar{x}`, rather than `\overline{x}`

## 6.4.4 Multiline Formulas

Multiline formulas ( or equations) or formulas displayed across multiple lines, are displayed with the `eqnarray` environment. It is very much like a *three-column array* environment, with consecutive rows separated by `\\` and consecutive items within a row separated by `&`. The default is to give an equation number for every line, unless that line has a `\nonumber` command typed just before the `\\`.

```
\begin{eqnarray}
q & = & q_c + q_r \nonumber \\
& = & h_c A(T_1 - T_2) + h_r A(T_1 - T_2) \\
& = & (h_c + h_r) A(T_1 - T_2)
\end{eqnarray}
```

$$\begin{aligned} q &= q_c + q_r \\ &= h_c A(T_1 - T_2) + h_r A(T_1 - T_2) \\ &= (h_c + h_r) A(T_1 - T_2) \end{aligned} \quad (6.3)$$

## 6.4.5 Mathematical Symbols

For the discerning user who wants to typeset complex mathematics,  $\LaTeX$  provides almost any mathematical symbol one is likely to need.  $\LaTeX$  range of letters for complex math notations includes upper- and lowercase Greek to uppercase calligraphic (invoked by the `\cal` command) letters. A complete list of these and other commands can be found in one of the  $\LaTeX$  manuals in the lab.

## 6.5 Doing More With $\LaTeX$

### 6.5.1 The tabular Environment

The `tabular` environment is used for making tables and can be used in any mode including math mode. It is similar to the `array` environment described in the previous section<sup>4</sup>, though in view of the way these two environments are processed by  $\LaTeX$  `tabular` is better suited for tabular lists and `array` for mathematical formulas.

In addition to the letters `c`, `l` or `r` specifying the positioning of the items in the columns, the argument of `tabular` could also have `|` characters, which puts a vertical line extending the full height of the environment in the specified place. Other features include the `\hline` command, which when specified after a `\\` or at the beginning of the environment, draws a horizontal line across the full width of the environment. The `\cline{i - j}` command draws a horizontal line across columns *i* through *j* inclusive. A single item that spans multiple columns is made with a `\multicolumn` command specified in the form

$$\backslash\text{multicolumn}\{n\}\{pos\}\{item\}$$

where *n* specifies the number of columns to be spanned, *pos* the horizontal positioning of the item, and *item* the item's text. Note that the *pos* argument replaces the portion of the environment's argument corresponding to the *n* spanned columns, and must contain a single `l`, `r` or `c` character; it may contain `|` characters also. Illustrated below is an example of a table created with the `tabular` environment.

---

<sup>4</sup>It pays to read the section on `arrays` before proceeding any further here.

```

\begin{tabular}{||1|c||} \cline{1-2}
\multicolumn{2}{||c||}{Crash of '87} &
\multicolumn{1}{1}{ } \\ \cline{1-2}
Date & Dow Index & \\ \cline{1-2}
Aug. 25 & 2722.42 & peak \\
Oct. 16 & 2246.73 & \\
Oct. 19 & 1738.74 & crash \\ \cline{1-2}
\end{tabular}

```

Crash of '87	
Date	Dow Index
Aug. 25	2722.42
Oct. 16	2246.73
Oct. 19	1738.74

The `tabular` environment produces an object that  $\text{T}_{\text{E}}\text{X}$  treats exactly like a single, big letter. You could put it in the middle of a paragraph, or even in the middle of a word. But for the sake of clarity, a table appears better when displayed between paragraphs, and for this purpose  $\text{\LaTeX}$  provides two environments which allow tables and similar objects (like pictures) to be displayed.

## 6.5.2 Tables and Figures

These two environments do not actually produce the figures or tables, instead they provide a means of displaying a figure or table in the most appropriate manner. The caption on a figure or table is made with a `\caption` command having the caption's text as its argument. The form of the environment is given below:

```

\begin{figure} body \end{figure}

```

The table created in the previous section could have been displayed as follows:

```

\begin{table}[h]
\begin{tabular}{||1|c||}... \end{tabular}
\caption{Dow's reflection of the crash.}
\end{table}

```

Both these environments have a counter attached which gets incremented every time the environment is used. For instance in the above case, let's say you were using the `table` environment for the third time in your document, the caption would be displayed as follows:

Table 3: Dow's reflection of the crash.

The `figure` or `table` is placed in with the text, usually just past the point where the figure or table is first mentioned. Though these environments are the same except for how they are captioned, the `figure` environment is generally used for pictures and the `table` environment is generally used for tabular information.

Another use of this environment is for leaving space for material to be pasted in later. This could be accomplished by using the `\vspace` command with its argument specifying the amount of vertical space, and the `\caption` command giving a caption for the picture or figure to be pasted in later.

```

\begin{figure}
\vspace{3.5in}
\caption{Caption for the figure}
\end{figure}

```

## 6.6 Preparing a Document

A document, like this primer for instance, consists of various chapters, sections, subsections and other lower level sectional units. This sectioning of the document enhances the clarity of presentation of the material.

In addition, the document, whether a book or a report should have a table of contents, list of figures etc, thus facilitating easier navigation through the document. L<sup>A</sup>T<sub>E</sub>X provides means for all these specifics, thus justifying the claim of being a document preparation system.

### 6.6.1 Sectioning Commands

As you must have observed in this primer, the various sections, subsections etc., have their sectional unit numbers according to the decimal system. This numbering was automatically done by L<sup>A</sup>T<sub>E</sub>X, with the counters associated with each section command. These same numbering and sectional headings may be used in constructing the table of contents and for producing the running head. The format for the sectioning command is as below:

```
sec-cmd[toc-entry]{heading}  
sec-cmd*{heading}
```

The *sec-cmd* could be one of the following:

```
\part      \section    \subsubsection  \subparagraph  
\chapter  \subsection  \paragraph
```

**Note:** *Each sectional unit must be contained in the next higher-level unit, though \part is optional, which is used mainly in book document style only. The article document style does not have a \chapter command. This primer used only the \section, \subsection and \subsubsection commands.*

The argument *toc-entry* produces the table of contents entry and may be used for the running head. If this optional argument is missing the *heading* argument is used for the above purposes. The *heading* argument produces the section heading.

The \*-form of the sectioning command suppresses the section number, does not increment the counter, does not affect the running head, and produces no table of contents entry.

### 6.6.2 Table of Contents

If the above sectioning commands are used in the preparation of the document, L<sup>A</sup>T<sub>E</sub>X will generate a table of contents by specifying just one command, which is `\tableofcontents`. The table of contents will be printed right at the point where the command appears, and hence it is customary to put this command at the beginning of the document itself. Two similar such commands are the `\listoffigures` and `\listoftables` which generate a list of figures, and list of tables, respectively. The entries of the above two lists are produced by the `\caption` command in the `figure` or `table` environment.

### 6.6.3 The Title Page and Abstract

You can create a title page with the `titlepage` environment. This environment creates a page with the empty pagestyle, so it has no printed page number or heading, and causes the following page to be numbered one. But it is completely up to the user as to the contents of this title page. You can design the title page in whatever form your imagination permits. When you are using the `article` document style you will have to explicitly declare `titlepage` in the *options* of the `\documentstyle` command.

An abstract is made with the `abstract` environment. The abstract is placed on a separate page in the `report` document style and when the `titlepage` style option is used with the `article` style; it acts like an ordinary *displayed*-paragraph environment with the plain `article` style. There is no `abstract` environment in the `book` document style.

## 6.7 Error Handling

An introduction to any software is incomplete without some mention about dealing with that inevitable evil — errors. Error handling in  $\LaTeX$  unfortunately is no comedy at all, and almost a chapter could be devoted to this topic alone. Since that is not practical considering the scope of this primer, a few pointers are included here which may prove helpful when faced with one of those rare gems – errors which only  $\TeX$  can produce.

There are two basic types of errors  $\LaTeX$  or  $\TeX$  flags you with. The first type involves those fatal or near fatal errors which requires the user’s involvement for further processing. When  $\LaTeX$  or  $\TeX$  faces such an error, it stops processing, writes some informative message and ends the message with a ?. The ? indicates that  $\LaTeX$  is waiting for you to help it out. The messages preceding the ? usually present the following information:

1. “**LaTeX error**”, indicating that the error was detected by  $\LaTeX$ . If this is missing, then it is a  $\TeX$  error.
2. An *error indicator*, which tells what the problem is. The error indicator message begins with an exclamation point (!).  $\LaTeX$  error indicator messages are easier to understand than those of  $\TeX$ .  $\TeX$  knows nothing about  $\LaTeX$  commands, so you can’t expect much help from the error indicator.
3. An *error locator*, which tells you where in your input file the error was discovered. The error locator starts with a line number such as 1.17, meaning that the error was found while  $\LaTeX$  was processing the seventeenth line from the beginning of the file. Then it prints a part of that particular line it processed before it encountered the error.

In most cases from the above information one can usually decipher what caused the error and then rectify it. Though not recommended, error rectification could be made at this level itself. A better approach is to stop the execution by typing `\stop`, or if that does not work by typing an `x` (which always works), and then making the change in the input file. However, the error messages which  $\LaTeX$  or  $\TeX$  provides you are very ambiguous and hence it is better to be a little more careful while creating the input file than to have to deal with those ambiguous errors later. Here are a few points (common mistakes) you should be careful of:

- Omitting braces around an argument or forgetting the `\` in a command name.
- Using preamble declarations or commands after the `\begin{document}` command.
- Using the reserved characters in ordinary text.
- Unmatched delimiters such as `{`, `\[`, `\(`, `$`, etc. This is one of the most common mistakes.
- Missing a `&` character or `\\` in an `array` or `tabular` environment.
- Using math-mode commands in paragraph mode.
- Forgetting to end an environment.

The second type involves warning messages which  $\LaTeX$  or  $\TeX$  gives concerning errors that are not fatal.  $\LaTeX$  does not stop processing in such cases, and hence no ‘?’ is printed. Examples include `Overfull \hbox ...` (when text goes beyond the right margin), etc., and could usually be ignored.

## 6.8 Formatting, Previewing and Printing

After you have written your document and saved it. Your filename must end with a *.tex* extension or you will not be able to view the document. For example, `primer.tex` is a valid L<sup>A</sup>T<sub>E</sub>X filename while `primer` is not. Use the `mv` command to rename your file if it does not have a valid filename.

In order to format your document, type `latex filename`<sup>5</sup> from the command line. If your document has no errors, then you will eventually see something like this:

```
(see the transcript file for additional information)
Output written on primer.dvi (72 pages, 179680 bytes).
Transcript written on primer.log.
```

You can now preview your document now by typing `xdvi filename &`. A window will pop up. Position the window with the mouse and click the left mouse button when ready. Your document will appear on the window. If you are not satisfied with the document, you can re-edit the file and repeat the process *until* you are satisfied. Because you have a print quota, this will be very useful. This way you do not have to print out a large file just to prove read it.

After you have typeset your document, printing it is no problem. The command `dvips filename` (**without the extension**) will print out a copy of your document. For example, `dvips primer` will convert `primer.dvi` into PostScript and send it to the printer.

A useful thing to know for the `dvips` command are the `-p` and `-l`. This allows you to specify which pages you want to print, e.g. you wish to print pages 55 through 63 (say it's a large document). Then you type `dvips -p 55 -l 63 primer`. Similarly, you can use the `-pp` option to specify which pages to print. Using this option, you specify specific pages (separated by a comma) or ranges of pages (where the range is given with a dash in the middle, as in a-b). Another useful `dvips` option is `-o <filename>`. Just as the `-o` option for the `gcc` compiler, this option writes the PostScript file to a file with the name specified by `<filename>` rather than to the printer. For more options, do a `man dvips`.

## 6.9 Other Things to Know

L<sup>A</sup>T<sub>E</sub>X can do so many different things that it truly is impossible to cover it all here. However, the following items are mentioned so that you are aware they *exist* and can be done if you need them. Look in a L<sup>A</sup>T<sub>E</sub>X manual for more information.

First, L<sup>A</sup>T<sub>E</sub>X has a `slides` environment. This is used for making slides for a presentation. You can create your slides using L<sup>A</sup>T<sub>E</sub>X (see a L<sup>A</sup>T<sub>E</sub>X manual for more on this) and then photocopy the printout onto transparencies.

Though L<sup>A</sup>T<sub>E</sub>X itself does just about anything you want to do, there may come a time when you need to use a L<sup>A</sup>T<sub>E</sub>X *package*. A package is like a L<sup>A</sup>T<sub>E</sub>X library with other commands not in L<sup>A</sup>T<sub>E</sub>X. To use a package, add the line

```
\usepackage{PackageName}
```

to your L<sup>A</sup>T<sub>E</sub>X preamble where *PackageName* is the name of the package you wish to include. For instance, to make your presentation (or paper) more striking, you may wish to use the `seminar` package or the `fancybox` packages in L<sup>A</sup>T<sub>E</sub>X. Then either the line `\usepackage{seminar}` or `\usepackage{fancybox}` (depending on which package you intend to use) must appear in your preamble.

---

<sup>5</sup>the `.tex` extension is not necessary

## Chapter 7

# Drawing Pictures and Plotting Graphs

### 7.1 Drawing using idraw

`idraw` and `xfig` are two utilities that allow you to draw pictures. Depending on which one you want to use, just type in `idraw &` or `xfig &` at your xterm prompt and a window will pop up.

We will concentrate on `idraw` in this chapter. (`man` pages for both `idraw` and `xfig` are on the Suns in the department.)

`idraw` is a drawing editor which allows you create, edit, re-edit and save drawings. Your drawing can consist of text, lines, splines, rectangles, polygons, and ellipses. You can create very intricate patterns with some practice and a lot of patience.

Since `idraw` allows you to add and delete objects, move them around, alter, duplicate and rotate them, you can create simple drawings very quickly.

The left mouse button is used for drawing objects or positioning text. The middle mouse button is used for selecting an already drawn object or moving it. The middle button also finalizes the lines, splines and polygons. The right mouse button is used to undo points selected while drawing lines, splines and polygons. All the keys can be used to select items from the menu.

Once you have created your picture, you can save it by typing `<<ctrl-s>>` or by selecting the `File` option and clicking the mouse on the appropriate choice. You can print the file directly to the laser printer by typing `<<p>>`.

You might want to include drawings which you created using `idraw` in your  $\LaTeX$  file. Luckily, `idraw` saves your files in PostScript format. This means that you can include your drawing in a  $\LaTeX$  file by including an additional option (the `psfig` option) in the  $\LaTeX$  document preamble where you specify the document class.

```
\documentclass[psfig,11pt]{report}
```

Wherever in the document you want to place the diagram, you should include something like this :

```
\hspace*{\fill}\psfig{figure=drawing.ps,height=7in,width=5in}\hspace*{\fill}
```

In this case the picture is read from a file named *drawing.ps*. The picture will be 7 inches in height by 5 inches in width.

You can also draw figures in  $\LaTeX$  – refer to the  $\LaTeX$  manual. However, it is usually much simpler to create a picture using `idraw` and include it into  $\LaTeX$  as discussed above.

## 7.2 Plotting graphs using `gnuplot`

### 7.2.1 Introduction to `gnuplot`

`gnuplot` is a Gnu tool to plot graphs and histograms. `gnuplot` has many features which make it useful. It has a variety of plot styles, and even does some kinds of 3D plots. It does confidence intervals. In addition, it produces postscript output.

To get started, type `gnuplot` at the Unix command line. Once you are in `gnuplot`, you will see a `gnuplot` prompt which looks like `gnuplot>`. You will use this prompt to type in all your `gnuplot` commands.

One of the first things you need to learn in `gnuplot` is how to get to the on-line help. At the `gnuplot` prompt, type `h`. Various help topics will appear on the screen. Select a topic and type it into the `gnuplot` prompt. You will now be presented with the on-line help information for the topic you chose. At the end of the help, they may be a list of sub-topics which are related to your topic. You can type in one of these sub-topics for help on more specific topics. Finally, to exit the on-line help, simply press `<<Return>>`.

To quit `gnuplot`, type either `quit` or `exit` at the `gnuplot` prompt.

### 7.2.2 plotting $(x, y)$ pairs

To plot some  $(x, y)$  pairs in `gnuplot`, you first need a file which contains the  $(x, y)$  pairs you wish to plot. This file has in it the  $(x, y)$  pairs, one pair on each line, with a blank space separating the  $x$  value from the  $y$  value. For instance, let's say you have a file `func.dat` which contains the data

```
1 1.1
2 2.2
3 3.3
4 4.4
```

To view this file in X-windows, at the `gnuplot` prompt, type `plot 'func.dat'`.

That was just to get started. Let's plot the same data, but make things a little more fancy. Now, instead of typing commands at the `gnuplot` prompt, let's put a bunch of commands in a 'command' file. For example, suppose the file `plot.com` contains:

```
# File: plot.com - to be used with func.dat
plot 'func.dat'
```

Just as in  $\LaTeX$ , any line beginning with `#` is a comment. Now, to see the results, you need to "load" this command file at the `gnuplot` prompt by typing `load 'plot.com'`. This produces the same plot as before.

To get even more fancy, let's "load" a file but this time you will write the output to a PostScript file rather than immediately to the screen. Create a file `"fig.ps"` in Unix which contains the following.

```
# gnuplot file for fig.ps
set size 0.73,1          # to make the picture square (gnu uses a rect)
set nokey                # don't want a key for the point shape
set noxtics              # don't want tick marks
```

```

set noytics
set format ""          # don't want the file name appearing
set output "fig.ps"   # output file
set terminal postscript portrait # a postscript file in portrait mode
plot [0:5] [0:6] "func.dat" with dots
pause -1 "Hit return to continue"

```

This creates a postscript file named “fig.ps” as output. Also, various improvements have been added to the plot.

- the plot box has been sized (set size)
- there’s no key (since we have only one line)
- no tick marks
- don’t want the file name “func.dat” appearing
- set the output to a file
- set the output to be postscript in portrait format
- define ranges for the axes ([0:5] for X, [0:6] for Y)
- uses only “dots” for points
- an option to continue is provided

The above will create a pretty spare plot (no symbols or tick marks).

### 7.2.3 Additional Features

The plot styles in `gnuplot` include

- dots - plain tiny dots (useful for scatter plots)
- points - special marks (diamonds etc) for points
- errorbars - used confidence intervals. The data file is either of the type (x,y,ydelta) or (x,y,ylow,yhigh)
- lines - lines that connect adjacent points
- linespoints - for both points and lines
- impulses - for vertical bars (like histograms)
- boxes - histograms
- boxerrorbars - histograms with confidence info
- steps - for step functions. This style connects consecutive points with two line segments: the first from (x1,y1) to (x2,y1) and the second from (x2,y1) to (x2,y2).

In order to use one of these plot styles, you would type something as in the following

- To plot with impulses (vertical bars), type `plot ‘‘func.dat’’ with impulses`

- To plot  $x * y$  with points,  $x * *2 + y * *2$  default, you would type `splot x*y w points, x**2 + y**2`
- To plot “exper.dat” with errorbars (confidence intervals) and lines connecting the points, type `plot 'exper.dat' with lines, 'exper.dat' with errorbars`. Here ‘exper.dat’ should have three or four data columns.
- To plot  $x**2 + y**2$  and  $x**2 - y**2$  with the same line type, `splot x**2 + y**2 with line 1, x**2 - y**2 with line 1`
- To plot file “data” with points style 3, type `plot "data" with points 1 3`
- To plot  $\sin(x)$  and  $\cos(x)$  with linespoints, using the same line type but different point types, type, `plot sin(x) with linesp 1 3, cos(x) with linesp 1 4`

Using `gnuplot`, you can also plot multiple plots on the same graph. Here’s an example of plotting two polygons on the same picture. Suppose the file “poly.dat” has one polygon and the file “poly2.dat” has another. To plot both, at the `gnuplot` prompt, type `plot "poly.dat" with lines, "poly2.dat" with lines`  
`gnuplot`, you can set the range for your graph if you wish it to be different from the default range. Here are some examples of various ranges you could set:

- To use the current ranges, type `plot cos(x)`
- To set the  $x$  range only, type `plot [-10:30] sin(pi*x)/(pi*x)`
- To set the  $x$  range only, but use  $t$  as a dummy variable, type `plot [t = -10 :30] sin(pi*t)/(pi*t)`
- To set both the  $x$  and  $y$  ranges, type `[-pi:pi] [-3:3] tan(x), 1/x`
- To set only the  $y$  range, and also to turn off autoscaling on both axes, type `plot [ ] [-2:sin(5)*-8] sin(x)**besj0(x)`
- To set  $x_{max}$  and  $y_{min}$  only, type `plot [:200] [-pi:] exp(sin(x))`
- To set the  $x$ ,  $y$ , and  $z$  ranges, type `splot [0:3] [1:4] [-1:1] x*y`

Another very important command is the “set” command. You use the “set” command to

- set terminal - sets the type of output (i.e. PostScript or  $\LaTeX$  output). For instance:
  - `set terminal postscript`
  - `set terminal postscript eps`
  - `set terminal postscript eps 14`
  - `set terminal postscript portrait`
  - `set terminal latex`
- set arrow - to draw an arrow
- set style - to change the plot style
- set size - to change size of plot
- set grid - draws a grid
- set label - to put string labels anywhere. For instance, `set label ‘quadratic’` at 1,2.5 will left justify the text at that point. `set label ‘y=x2’` at 1,2.5 center will center the label at the point (1,2.5)

- `set xlabel` - to put a label on the x-axis, e.g., `set xlabel 'temperature'`

In the `'plot'` command, blank lines in the data file cause a break in the plot. There will be no line drawn between the preceding and following points if the plot style is `'lines'` or `'linespoints'` (see `'plot style'`). This does not change the plot style, as would plotting the data as separate curves. To specify other formats, see `'plot datafile using'`.

The following example compares the data in the file `population.dat` to a theoretical curve:

```
pop(x) = 103*exp((1965-x)/10)
plot [1960:1990] 'population.dat', pop(x)
```

Where file `population.dat` might contain:

```
# Gnu population in Antarctica since 1965
1965  103
1970   55
1975   34
1980   24
1985   10
```

When a data file is plotted, `'samples'` and `'isosamples'` are ignored. Curves plotted using the `'plot'` command are automatically extended to hold the entire curve. Similarly, grid data plotted using the `'splot'` command is automatically extended, using the assumption that isolines are separated by blank lines (a line with only a CR/LF in it).

Implicitly, there are two types of 3-d datafiles. If all the isolines are of the same length, the data is assumed to be a grid data, i.e., the data has a grid topology. Cross isolines in the other parametric direction (the *i*th cross isoline passes through the *i*th point of all the provided isolines) will also be drawn for grid data. (Note contouring is available for grid data only.) If all the isolines are not of the same length, no cross isolines will be drawn and contouring that data is impossible.

For `splot`, data files may contain more than one mesh and by default all meshes are plotted. Meshes are separated from each other, in the file, by double blank lines. To control and plot a single mesh from a multi mesh file, use the index modifier. See `'splot index'` for more.

For `splot` if 3-d datafile and using format (see `'splot datafile using'`) specify only *z* (height field), a non parametric mode must be specified. If, on the other hand, *x*, *y*, and *z* are all specified, a parametric mode should be selected (see `'set parametric'`) since data is defining a parametric surface.

A simple example of plotting a 3-d data file is

```
set parametric
splot 'glass.dat'
```

or

```
set noparametric
splot 'datafile.dat'
```

where the file `datafile.dat` might contain:

```
# The valley of the Gnu.
10
```

```

10
10

10
5
10

10
1
10

10
0
10

```

Note datafile.dat defines a 4 by 3 grid ( 4 rows of 3 points each ). Rows are separated by blank lines.

On some computer systems with a popen function (Unix), the datafile can be piped through a shell command by starting the file name with a '<'. For example:

```

pop(x) = 103*exp(-x/10)
plot '< awk "{print $1-1965, $2}" population.dat', pop(x)

```

would plot the same information as the first population example but with years since 1965 as the *x* axis. If you want to execute this example, you have to delete all comments from the data file above or substitute the following command for the first part of the command above (the part up to the comma):

```

plot '< awk "$0 !~ /^#/ {print $1-1965, $2}" population.dat'

```

It is also possible to apply a single function to the *y* value only, e.g.

```

plot 'population.dat' thru p(x)

```

For more information about 3-d plotting, see 'splot'.

The format of data within a file can be selected with the 'using' option. An explicit scanf string can be used, or simpler column choices can be made. The syntax is:

```

plot "datafile" { using { <ycol> |
                        <xcol>:<ycol> |
                        <xcol>:<ycol>:<ydelta> |
                        <xcol>:<ycol>:<ylow>:<yhigh> |
                        <xcol>:<ycol>:<ylow>:<yhigh>:<boxwidth> }
                        {"<scanf string>"} } ...

```

and

```

splot "datafile" { using { <xcol>:<ycol>:<zcol> | <zcol> }
                        {"<scanf string>"} } ...

```

<xcol>, <ycol> and <zcol> explicitly select the columns to plot from a space or tab separated multicolumn data file. If only <ycol> is selected for 'plot', <xcol> defaults to 1. If only <zcol> is selected for 'splot',

then only that column is read from the file. An `<xcol>` of 0 forces `<ycol>` to be plotted versus its coordinate number. `<xcol>`, `<ycol>` and `<zcol>` can be entered as constants or expressions.

If errorbars (see also ‘plot errorbars’) are used for ‘plot’s, `ydelta` (for example, a +/- error) should be provided as the third column, or `ylo` and `yhigh` as third and fourth columns.

If boxes or boxerrorbars are used for ‘plot’s, a fifth column to specify the width of the box may be given. This implies that columns three and four must also be provided even if they are not used. If you want to plot boxes from a data file with three columns, set `ylo` and `yhigh` to `y` using the following command:

```
plot "datafile" using 1:2:2:2:3 with boxes
```

Scanf strings override any `<xcol>:<ycol>(:<zcol>)` choices, except for ordering of input, e.g.,

```
plot "datafile" using 2:1 "%f%*f%f"
```

causes the first column to be `y` and the third column to be `x`.

If the scanf string is omitted, the default is generated based on the `i<xcol>;i<ycol>(:i<zcol>)` choices. If the ‘using’ option is omitted, `"%f%f"` is used for ‘plot’ (`"%f%f%f%f"` for ‘errorbars’ ‘plot’s) and `"%f%f%f"` is used for ‘splot’. For example,

```
plot "MyData" using "%f%f%*20[^\n]%f" with lines
```

Data are read from the file “MyData” using the format `"%f%f%*20[^\n]%f"`. The meaning of this format is: “first number, `"%*20[^\n]"` then ignore 20 non-newline characters, `"%f"` then read in the `y` value. For instance,

```
n=3;
plot "MyData", "MyData" using n
```

causes gnuplot to plot the second and third columns of MyData versus the first column. The command `'n=4; replot'` would then plot the second and fourth columns of MyData versus the first column.

```
splot "glass.dat" using 1
```

causes gnuplot to plot the first coordinate of the points of glass.dat as the `z` coordinate while ignoring the other two coordinates. Note: gnuplot first reads a line of the data file into a buffer and then does a

```
sscanf(input_buffer, scanf_string, &x, &y{, &z});
```

where ‘`x`’, ‘`y`’, and ‘`z`’ are of type ‘float’. Any scanf string that specifies two (three for ‘splot’, three or four for ‘errorbars’) float numbers may be used.

You can also include plots and graphs which you created using `gnuplot` in a  $\LaTeX$  file. You need to save your plot in  $\LaTeX$  format using the “set terminal” command. Wherever in the document you want to place your graph or chart, you should include something like this:

```
\hspace*{\fill}\input{plot.tex}\hspace*{\fill}
```

# Appendix A

## Unix Commands

A list of some more useful Unix commands together with a terse description. To learn how to use the command, look up the corresponding `man` pages. They are listed in alphabetical order for your viewing pleasure. Hopefully you will be encouraged to go through the system yourself and find other handy commands or utilities.

`alias` - to set a substitute (and a more convenient) name for a command  
`apropos` - locate commands by keyword lookup  
`bb` - bulletin board announcement utility  
`bc` - arbitrary-precision arithmetic language  
`cal` - display a calendar  
`cat` - concatenate and display  
`cb` - C program beautifier  
`cc` - C compiler (K&R)  
`cd` - change directory  
`chmod` - change the permissions mode of a file  
`cmp` - compare 2 files  
`compress` - compress a file.  
`crypt` - encode or decode a file  
`csh` - a shell (command interpreter) with a C-like syntax and advanced interactive features  
`ctrace` - generate a C program execution trace  
`dbx` - source-level debugger  
`dclock` - digital clock for X  
`df` - show how much free disk space there is  
`diff` - display line-by-line differences between two text files  
`du` - display the number of disk blocks used per directory or file  
`dvips` - convert a  $\text{T}\text{E}\text{X}$  or  $\text{L}\text{A}\text{T}\text{E}\text{X}$  DVI file to PostScript  
`emacs` - GNU project Emacs.  
`f77` - fortran 77 compiler  
`find` - search for a file  
`finger` - display information about users

`ftp` - file transfer program  
`gcc` - GNU project C Compiler. Supports all ANSI standard C programs.  
`gs` - PostScript previewer  
`gprof` - to measure performance statistics of your program  
`grep` - search a file for a pattern  
`history` - display the history list, to avoid re-typing entire commands.  
`hostname` - which host are you on  
`id` - definitions of user, group  
`idraw` - drawing edit  
`kcl` - Kyoto Common Lisp  
`kill` - send a signal to a process, or terminate a process  
`last` - record of logins  
`latex` - typesetter  
`leave` - a reminder program  
`lex` - lexical analysis program generator  
`lint` - type checker for C  
`less` - view a file with options to move forward or backward  
`lpq` - display the queue of printer jobs  
`lpr` - send a job to the printer  
`lprm` - remove jobs from the printer queue  
`ls` - list the contents of a directory  
`mail` - read or send mail messages  
`make` - compile, link etc according to dates files are modified  
`man` - display reference manual pages; find reference pages by keyword  
`mkdir` - make a directory  
`more` - browse through a text file. Also see `less`!  
`mv` - move or rename file  
`oclock` - analog clock for X  
`passwd` - change local or NIS password information  
`pc` - the Pascal compiler  
`plotxy` - program for generating graphs from data files  
`ps` - check id, status etc., of processes running on system  
`pwd` - what is the current directory  
`quota` - disk space quota  
`rlogin` - establishes a remote login session from your terminal to the remote machine you specify  
`rm` - remove (unlink) files  
`rmdir` - remove (unlink) directories  
`rsh` - remote shell. Connects to the specified hostname and executes the specified command.  
`screenload` - load a frame-buffer image from a file. `screenload` reads a Sun standard rasterfile.  
`sh` - shell, the standard UNIX system command interpreter and command-level language  
`source` - to force acceptance of unix commands from a file  
`spell` - report spelling errors

**tail** - display the last part of a file  
**tar** - retrieve files from archives  
**tcsh** - C shell with file name completion and command line editing  
**telnet** - login to some other machine (on internet)  
**tex** - text formatting and typesetting  
**tty** - terminal characteristics. See also **stty**  
**ups** - X11 and SunView based source level C debugger  
**vacation** - to set up an automatic mail reply  
**vi** - visual display editor  
**who** - who is logged in on the system  
**xcalc** - scientific calculator for X  
**xclock** - analog / digital clock for X  
**xdbx** - X window system interface to the dbx debugger  
**xfig** - Facility for Interactive Generation of figures under X11  
**xloadimage** - load images into an X11 window or onto the root window  
**xmail** - X11 visual interface to the mail program  
**xman** - Manual page display program for the X Window System  
**xmh** - send and read mail with an X interface to MH  
**xplot** - plot filter for X  
**xterm** - terminal emulator for X  
**xvbb** - Graphical User Interfaces (GUIs) for bb  
**yacc** - yet another compiler-compiler: parsing program generator

# Appendix B

## Key bindings

A list of key bindings used by GNU Emacs. They are ordered by function. Here C-x denotes `<<ctrl-x>>`.  
Ordered by function:

Movement:	C-f	forward-char	
	C-b	backward-char	
	C-p	previous-line	
	C-n	next-line	
	C-a	beginning-of-line	
	C-e	end-of-line	
	C-v	scroll-up	
ESC-v	scroll-down		
Deleting text:	DEL	delete-backward-char	
	C-d	delete-char	
	ESC-d	delete-word	
	C-k	kill-line	
Using regions:	C-@	set-mark-command	
	C-w	kill-region	
	C-y	yank	
Searching:	C-s	isearch-forward	
	C-r	isearch-backward	
	ESC-%	query-replace-string	
File handling:	C-x C-s	save-buffer	
	C-x C-c	save-buffers-kill-emacs	
	C-x C-f	find-file	
	C-x i	insert-file	
Windows:	C-x 2	split-window-vertically	
	C-x 1	delete-other-window	
	C-x o	other-window	
Miscellaneous:	C-g	keyboard-quit	(quit current command)
	C-l	recenter	(refresh screen)
	C-o	open-line	(open line to enter text)
	C-h	help-command	(general help)
	C-h t	help-command	(tutorial)

# Appendix C

## Preamble for L<sup>A</sup>T<sub>E</sub>X

`\documentstyle`

This declaration is used to specify the document style, the arguments of which choose one of the predefined styles. The form of the declaration is as given below:

`\documentstyle[option1,...,optionk]{style}`

*style* The main document style; the standard<sup>1</sup> ones are: `article`, `report`, `book`, and `letter` (for letters only). There is also a `slides` style for use only with S<sup>L</sup>T<sub>E</sub>X. The `\documentstyle` command reads the file *style.sty*.

*options* A list of one or more style options, separated by commas—with no spaces. The standard L<sup>A</sup>T<sub>E</sub>X options are:

- `11pt` Makes eleven-point type the normal (default) type size instead of ten-point type.
- `12pt` Makes twelve-point the normal (default) type size instead of ten-point type.
- `twoside` *formats* the output for printing on both sides of a page.
- `twocolumn` Produces two-column pages.
- `titlepage` Produces separate title page for `article` style.
- `leqno` Puts formula numbers on left side in `equations` and `eqnarray` environments.
- `fleqn` Left-aligns displayed formulas.

If you don't like the standard L<sup>A</sup>T<sub>E</sub>X document styles, you can create your own. See the L<sup>A</sup>T<sub>E</sub>X manual for this.

`\pagestyle{style}`

This declaration specifies the current page style. An output page consists of a *head*, a *body*, and a *foot*. The page style specifies the contents of the head and the foot. The standard style options are

- `plain` The head is empty, the foot has only a page number. It is the default page style.
- `empty` The head and foot are both empty.

- **headings** The head contains information determined by the document style (usually a sectional-unit heading) and the page number; the foot is empty.
- **myheadings** Similar to the **headings** page style, except you specify the information that goes in the head, using the `\markboth` and `\markright` commands described below.

```
\markboth{left-head}{right-head}
\markright{right-head}
```

The *left-head* and *right-head* arguments specify the information to go in the page heads of left-hand and right-hand pages, respectively. In one-sided printing, all pages are considered to be right-hand ones.

`\parindent=`

Specifies the width of the indentation at the beginning of a paragraph. Its value may be changed anywhere in the document. By declaring this in the preamble, the whole document is set to the same indentation. The dimensional unit could be any of the units of length recognized by L<sup>A</sup>T<sub>E</sub>X viz.,

```
cm Centimeters.
em One em is about the width of the letter M in the current font.
ex One ex is about the height of the letter x in the current font.
in Inches.
pc Picas (1pc = 12pt).
pt Points (1in = 72.27pt).
mm Millimeters.
```

`\parskip=`

Specifies the extra vertical space inserted before a paragraph. Like the `\parindent` command its value may also be changed anywhere in the document.

### Other preamble declarations

```
\pagenumbering{num-style}
```

Specifies the style of page numbers. This also could be changed anywhere in the document. In the absence of this command, L<sup>A</sup>T<sub>E</sub>X chooses `arabic` as the default style. Possible values of *num-style* are:

```
arabic Arabic numerals.
roman Lowercase Roman numerals.
Roman Uppercase Roman numerals.
alph Lowercase letters.
Alph Uppercase letters.
```

### Style Parameters

The default values of the style parameters are changed in the preamble. Some of these parameters are given below<sup>2</sup>:

```
\oddsidemargin One inch less than the distance from the left edge of the paper to the left margin of the text on right-hand pages.
```

```
\evensidemargin The same as \oddsidemargin except for left-hand pages.
```

```
\topmargin One inch less than the distance from the top edge of the paper to the top of the page's head.
```

```
\headheight The height of (a box containing) the head.
```

```
\headsep The amount of vertical space between the head and the body of a page.
```

---

<sup>2</sup>Refer to the style file `artadj.sty` to see how the values for these parameters are specified.

`\textheight` The normal height of the body of a page.  
`\textwidth` The normal width of the text on the page.  
`\topskip` The minimum distance from the top of the body to the bottom of the first line of text.  
`\footheight` The height of (a box containing) the page's foot.  
`\footskip` The distance from the bottom of the last line of text in the body of a page to the bottom of the foot.