A Student's Guide to CDF

Welcome to CDF

Computer Science courses offered on the St. George campus above the introductory level do their computing on CDF, the Computing Disciplines Facility. CDF sometimes also supports courses in Electrical and Computer Engineering.

CDF is based on the Unix operating system, and consists of a network of workstations, together with several servers providing file storage and additional computing power. The Department of Computer Science is grateful to both the University's administration and various donors for their generosity in suporting computing education.

Where is it?

All the CDF workstations are located in the Bahen Centre for Information Technologies, 40 St George St. The labs are located on the second and third floors in the south wing (above the College St. entrance). Access is available 24 hours a day seven days a week by the use of your student card (Tcard).

The Gerstein Science Information Centre (9 King's College Circle) contains workstations running the Windows operating system. Those machines are used primarily by introductory computer science courses, and are not discussed in this guide. See the CDF-PC guide for information regarding the CDF-PC system

How to read this handbook

This guide assumes you, the reader, are a student in a Computer Science course using CDF. While primarily intended to help those new to CDF, even students who have been here a while might be reminded of useful things by glancing through this handbook. And most likely, as a student new to CDF, your most recent experience will have been on CDF-PC, the computing facility supporting introductory courses in Computer Science; so we will refer to what you learned there to make the explanations here a little shorter.

Because of this background, there are three major chapters to this handbook:

- **Getting Started** a collection of important information about what you can do at CDF, what your responsibilities are, and how to learn more details.
- **Finding your Way** information for new users.
- **Elementary Unix** an introduction for people used to a Windows or MS-DOS–like system;

Change is constant.

At the time of writing, the Computer Science Department is experiencing major changes connected with the completion of the new Bahen Centre, and concurrent changes in the way user accounts are structured at CDF so this handbook can't be totally correct. You may hear about changes from messages on the system, or by simply noticing the new behaviour of a command or the existence of a new icon.

Other sources of help

This handbook can only give brief advice about Unix, X Window and CDF, and you will soon want more details. One place to start is by asking your fellow students. Those around you will know the answers to many of your questions, and they won't mind answering: someone answered their questions once upon a time. You can ask the CSSU too, (http://www.cdf.toronto.edu/~cssu/); they're a sort of formalization of asking your friends. Check the newsgroups, either for your course or ut.cdf.general for general advice. If you have a technical problem that you can't resolve by reading manual pages or asking informally for advice, you may ask the sysadmins, by e-mail to "admin@cdf.toronto.edu".

But before you send your question, try the *faq* command and the CDF home page.

Rules

CDF is a shared facility: as a user, you work in the electronic and physical presence of many others. The community of our users is so large and diverse that we now set out some basic rules governing your behaviour. Below is a summary of these rules. The rules are discussed in greater detail later in this document.

Rules for CDF users

This summary is posted regularly in the electronic news.

- You are totally responsible for how your account is used.
- Account sharing is forbidden. You must not use anyone else's account, and you must not allow anyone else to use your account.
- Your account must not be used to disrupt the intended functioning of CDF or any other computing system.
- Your account must not be used in a way that a reasonable person would find threatening or offensive. This applies to audible output, screen displays, personal web pages and messages sent by email or to newsgroups.
- You must not use your account to attempt entry to accounts of which you are not a legitimate user, on this or any other computing system.
- You must use your account only for its intended purpose—generally, course work in the course for which it was issued. Program accounts are also to further your general education in computer science.

  However, you may use your account for non-educational purposes such as playing games, provided that such activities do not interfere with other users' legitimate uses of CDF. Read the rules about game playing before playing.
- The files associated with your account—including your mail—belong to the University of Toronto.

The University's Code of Behaviour on Academic Matters and the Code of Student Conduct govern what happens when you break these rules. Activities that seem likely to threaten system security or interfere with other users may result in immediate suspension of your account.

For more on these rules, check these sources:

- *A Student's Guide to CDF* [which is what you're now reading];
- these commands:

`rules`	(The official CDF rules);
`rules -x`	(Some philosophical background);
`rules -s`	(A discussion of safety and security issues);
`rules -g`	(Rules for game-players);

- the CDF frequently-asked-questions facility—the "faq" command;
- the CDF home page, www.cdf.utoronto.ca.
- University's Code of Behaviour on Academic Matters and the Code of Student Conduct found in the Arts & Science calendar

Getting started

In this section we cover important aspects of CDF, what you can do at CDF, what your responsibilities are, how to learn more details, and we try to point out areas and topics that we have found trip up or confuse new students.

Logging in at CDF is very similar to logging in at CDF-PC or to any other system requiring an account name and password. Here are some things to remember:

- **You're sharing the system**. Your files are kept on a large hard disk, along with all other users' files. When you're in one of the labs you'll see some of the other users around you, but even if you log in from a remote location you are sharing the system with the other users. Try not to make their lives too difficult.
- **You must NOT turn the workstation off**. Even if a workstation appears to be malfunctioning, do not turn it off. If there seems to be a problem with the workstation you want to use, contact the system administrators (admin@cdf, or the offices in BA3245) but don't turn the computer off or try to reboot it. Don't turn it off when you leave either, just log out.
- **You must never share your password with anyone, ever**. Your initial password is your student number. The first time you log in, you will be reminded of the rules, and then required to choose a new password. See below for advice on making this important choice.
- **Starting September 2002 there is a new account naming scheme**. You may get old information from other sources that have not been updated. User accounts will be of the form c2llllll (where l is a letter). See below for how to determine what you account name will be.

Good citizenship

CDF is a shared system. Although you are alone when you are logged in to a workstation, your files share space with others' files, your output goes to printers shared with other users, and as a person you share with other people the various resources such as the rooms and the workstations themselves.

In order for this arrangement to work in a fair and friendly way, all users have certain responsibilities. You must try not to use more than your share of time, space and paper. You must set up your account so that illicit users would find it difficult to break in, and you must be attentive to what is happening so that misbehaviour by others will be detected before serious damage occurs, either to the physical machinery or to the files and software. You must not eat or drink in the workstation rooms, nor litter. Please clean up papers or debris left at the workstations by less conscientious users.

You may think it unreasonable to ask students to take responsibility for a university-owned system like CDF. But it is impossible to station official watchers in every room or to chain down all removables; and you wouldn't like it anyway if we could do that.

We do not ask you to be a police officer. We ask only that you behave like an older sister or brother. Assert your own rights, but don't over-assert them. Be conscious of others' needs and help to educate the less mature users: most misbehaviour by users is caused by ignorance, not malice.

The next sections give more detail on these two major aspects of good behaviour: sharing and security. There are special sections on sharing enforcement and on choosing a good password, an important aspect of security.

You should also read the "rules", posted in on-line news regularly, and conveniently accessible with the command *rules*.

How to share

Here are some rules that will make it easier to share resources:

- Don't write your programs at the workstation. Plan ahead, so as to spend your time editing and running programs.
- Be considerate of other users. Computing needs concentration, and people are likely to be easily distracted when they're tired or worried about finishing their work.
- Don't talk loudly with your friends in the workstation rooms. If you want to carry on a conversation, find a more appropriate location.
- If you have time to play computer games, make sure they don't make sounds audible to other users.
- The CDF policy on game playing is posted often in the newsgroup "ut.cdf.announce". Read it and obey the rules! Among other things, it instructs you *never* to play games unless some workstations are free in the same room.
- Don't display images that others are likely to find offensive or threatening.
- Use headphones with personal stereos and reduce the volume so only you can hear it.
- Read newsgroups or email without spending too long at it.
- Try to do your assignments ahead of time. The machines will be *very* busy the night before your due date. In fact, they'll be busy just before any course's due date; use the command *deadlines* to find out when other courses have assignments due.
- Don't print very big files unless you really have to. For example, if you've made only a few changes in your program since the last printed version, don't print it again. Use tools to reformat lecture notes or other documents so that they use fewer pages. See the command *psnup* for ways to do this.
- If you do need a printout of a big file, use the command *lpq* to find out how busy the printer is; consider doing your printing later on if a lot of small jobs are being done now, or find a printer in another room that is idle. Use the command *printers* to find the names of other printers.
- Try not to keep big files that you don't need. Obvious files you might delete are compiler outputs for programs not in use—especially if you also have the source files—and execution outputs that you've already printed. Spend some time occasionally wondering whether you need all those files. See the commands *zip* and *gzip* for ways to compress files to take up less disk space.
- Don't keep multiple copies of source or data files, apart from perhaps a single backup copy of vital files. Read up on *rcs* and *cvs* for ways to keep track of previous versions of your program.
- This may seem irrelevant to sharing and security, but awareness of what's going on will help you cooperate with other users. Socialise with other students, attend events, and don't spend all your time working.

Limits and quotas

Three resources are important enough that sharing may need to be enforced: file space, printer output, and the workstations themselves.

File space

Your files live on a central computer, a "file server" that is shared with all other users. You have a *disk quota* or limit that indicates how much space your files are allowed to take up. Your quota is likely to be at least 15 megabytes.

If you exceed this quota or "soft limit" you are notified during login, and you should remove as many files as you can. Obvious candidates for deletion are "core" files from program failures, debugging output listings, executable files, and your own backup copies. You should also consider moving files onto floppy disks. Read "man floppy" to find out how to use floppy disks on the workstations.

If you exceed the “hard limit”—about 10% more than the soft limit—or if you have been over the soft limit for too long, you can no longer write files to disk. You can fix this *only* by deleting files. If you exceed the hard limit while editing, you will be unable to save the work you’re doing in your own home directory. Instead, save it in the “temporary” directory */tmp*, giving it a name like /tmp/c1abcdef if c1abcdef is your login id. Then delete unnecessary files and copy the temporary file to your home directory. You have to do this promptly, because files in */tmp* are destroyed irrevocably once or twice a day.

The error message you’ll see when your hard limit is exceeded is “No space left on device.” This is wrong—it’s not the whole device that’s full—but the wrong error message is Unix’s fault, not ours, so we can’t fix it.

The command *quota* will tell you how much disk space you’re using. It reports your usage on */homes*, where your home directory is kept, and on */mail*, where your mail is. You will probably find you are nearly always well below your limits, which are intended to prevent the few irresponsible users from completely filling the disk, as could sometimes happen.

Printer output

Printing is expensive. To save paper and to reduce costs, the amount of printing you can do for free is limited: you are allowed 300 pages per course, though some courses may receive less than that if they don’t have heavy printing requirements. A “page” is one side of a piece of paper. The command *pquota* will tell you how much printing you’ve done and how much of your quota is left.

This print limit is not so stringent as it sounds, because you can print more than one “page” of output on one physical side of a piece of paper. The command *print* can help you use your quota efficiently in this way. Read about it with “man print”.

You should print only what you really need to print in order to keep within your quota; almost all students do manage to finish their coursework in this way. However, if you reach your limit with printing still left to do, visit the system administrators in BA 3245 to pay the fee for further output. The fee is not enormous, but you can keep from paying it if you print conservatively.

Workstations

At busy times, you may find yourself waiting to use a workstation. If that becomes frequent, we will consider mechanisms to limit the length of users’ sessions.

Without limiting your *working* time, CDF already limits the length of time you can leave a workstation idle. A “daemon” regularly inspects workstations that have users logged in, and logs off those showing no activity for a period between ten minutes (when the labs are busy) to an hour.

Security

These rules are intended to avert serious damage to the computers and the information stored there. Violations are viewed as serious.

- Don’t eat, drink or smoke in the labs, or leave paper or other garbage lying around.
- Watch for people who seem to be tampering with the equipment. We will have the best physical security we can manage consistently with good service to users, but it won’t be perfect. Keep an eye open: those are *your* computers.
- Keep your files private. Use *chmod* to set your permissions so that they aren’t readable when they shouldn’t be. Your *umask*, probably set in your **.**login file, determines the default permissions on your newly-created files. Set it to 77; “**`man umask`**” will tell you about it.

  If someone else copies your files for illegal purposes, and it turns out that you left them unprotected, you will be held partly responsible.
- Choose a secure password; see the next section.
- Be aware of your files; if they change surprisingly, or if you have other evidence of tampering with your account, notify the system administrators by sending mail to “admin”.

Choosing a password

- Keep your password secret. You're responsible for everything done with your account, and account sharing is forbidden. Don't tell your password to *anyone*— good advice is to treat it like your toothbrush; don't share it and change it every six months.
- Make your password hard to guess. Don't use words or names; do use at least 7 characters—8 is the maximum—including upper-case letters, digits and/or special characters as well as lower-case letters. You'll find that the *passwd* command enforces some of these requirements.

  Your password should also be hard for someone who knows you to guess. A simple rule is that you shouldn't use any word or a name that could be found in your wallet, backpack or briefcase.
- Don't write your password down.

Penalties

At CDF, our basic approach is that we are trying to run a system supporting many users, and actions indicating that you are likely to interfere with that goal may require us to suspend your account. This approach and some of its corollaries are discussed in our regular news postings and can also be read by running the *rules* commands.

The University as a whole also has rules governing computing offences. There are in fact two codes: the Code of Behaviour on Academic Matters, and the non-academic Code of Student Conduct. Sometimes it may be a little unclear *which* code is relevant, but it is very clear that *some* code will apply to an offence on CDF. And the University considers that it has jurisdiction even if the misbehaviour actually took place at a distant site and CDF was involved only by providing network access.

Academic offences

Sometimes, on the other hand, we see misbehaviour on CDF that is primarily academic rather than computing-related. Such cases usually involve plagiarism or other forms of cheating, but they can also arise when your actions interfere with someone else's attempts to complete coursework.

The University asks us to publicize this warning:

> **Any attempt to interfere with the operation of a computing system or with the work of a fellow-student may be looked upon as a serious breach of academic discipline. Such activities as interfering with others' files, changing their passwords or copying their programs could lead to an appearance before the University Tribunal and to a punishment ranging from 0 in the course to expulsion from the University.**

Among other things, you should realize that copying programs, like copying essays, is a form of plagiarism.

It is also worth mentioning that your computing account has been given to you for use solely by you. Its use by others or for other than the intended purposes—for example, course work in courses not supported by CDF, or work related to your employment—may result in its suspension or cancellation.

Disaster Recovery

Deleting files

When a file is deleted on a Unix system there is not a handy container from which to retrieve the deleted file. This is guarantied to cause you anguish at some point during your time here. The degree of grief you experience can be moderated by precautions you take beforehand. See the man pages for rcs and cvs for methods of doing version control and rescuing yourself from this fate. The time spent learning this will pay itself off many times.

Backups

Regular backups of CDF files are done by the system administrators. These backups are really intended to guard against disk or system failures, but they can sometimes be helpful in case of user errors.

Some day, you will make a horrible mistake and delete or seriously damage one or many of your files. You then have two choices: do a lot of typing, or try to find an old “backed up” copy of the files. Send mail to admin@cdf to request a file restoration from backup, indicating the file or directory name and path but be aware the backups will be hours or days old.

Restoring system defaults

Invisible “dot” files control the behaviour of many components of your working environment. These files aren’t really invisible, but files whose first character is a period aren’t displayed in a default directory listing. Generally they are files you don’t want to be bothered with, such as configuration files that determine how your account will behave, and command files that are run by specific programs when they start up.

You can examine and change them if you wish. If you look in your “**.**login” file, using a text editor as described in the Introductory Unix section, you’ll see various commands determining the initial appearance of the screen, just after you log in. You can modify these in fairly obvious ways to make the screen more comfortable for you.

Occasionally by accident or misadventure you will mess up your **.**login, **.**cshrc, or some other file. When that happens you can give the command

```
/local/bin/startupfiles
```

to restore the system defaults.

Finding your way around the CDF system

Getting your account

Course accounts

Ordinarily, an account will be set up for you the first time you take a course that uses CDF. Your name and student number are obtained from the official lists of students enrolled in courses, and you receive an account with a name something like

`c2clarke`

Here the “c” indicates that this is a course account. The “2” indicates the last digit of the year the account was created (eg 2002). The “clarke” is the first six characters of your family name, which we suppose is “Clarke”; if there are two Clarkes then you might get “clarkf” instead of “clarke”, and the next Clarke would get “clarkg”. If your family name has fewer than six characters, some of your given name will be used. For example, Bill Joy would be “c2joywil” if his student card calls him William.

Your account will be used for all your St George campus CSC courses that use CDF for the remainder of your time at the University, as long as you continue to take at least one such course per academic year. However the account will only be active during the time you are enrolled in a course; when the course ends the account will be suspended. Your files, password and any personalization of the account will remain preserved until the next time you take a course, at which time the account will be reactivated. Note however that if an entire academic year passes without a course being taken the account will be deleted.

The printing limit is initially zero, and then is set to the quota for **one** of the courses you are enrolled in. If you are in more than one CDF course, you can increase your print quota by the use of the *pquota* command with the ‘*-i*’ (for increment) option: ‘ *pquota –i* ’.

One last point: **your initial password is your student number**. When you first log in, you will be forced to change it. Too many people know your name and number for the password to be left unchanged. It is important to choose a secure password, since your account will be around for a long time.

Program accounts

If you are enrolled in one of our undergraduate programs, then you may request a different account, instead of the course account you first received. The main advantage of having a program account is you can use the account for work relevant to your education in computer science even at times when you happen not to be taking any courses that require the use of CDF. Course accounts are suspended during periods the owner is not enrolled in a course.

A social advantage of a program account is that it gives you a kind of identity that we hope will make you feel more at home at CDF. Besides its continuous availability, your account has a name of your choosing. g1joe might be Joe Clarke’s account name. Here the “g” indicates the St. George campus and the “1” is the last digit of the year in which the account was set up. The remaining characters were Joe’s choice.

In return we expect program account holders to accept some responsibility. You should be a little more aware of correct computing behaviour, and more willing to help less experienced users. During terms when you are not taking any courses on CDF, you should log in regularly to check your mail and make sure that your files are as you left them and that there are no signs of intrusion into your account. (If you expect not to be using CDF for a long while or over the summer consider using the ‘*suspendme*’ command). And it would be nice of you, if you are not enrolled currently in any CDF courses, to try to avoid occupying a workstation when CDF is very busy. The primary purpose of the system, after all, is to support active course work.

You can request a program account online at http://www.cdf.toronto.edu/permanentaccounts/

If you are not in a Major or Specialist undergraduate CSC program in the Faculty of Arts and Science on the St. George campus, then we cannot arrange a program account for you. Accounts are not offered to students in the Minor program, to students in Engineering, or at UT Mississauga or UT Scarborough, all of which offer their own computing support to their students.

Interaction with the system

To begin using a CDF workstation, you first log in, providing your account name and password in the usual way. Your password can be changed from a command prompt using the command "**`passwd`**", and you should change it if you ever suspect you might have revealed it. Nobody but you should know your password—not even your mother!

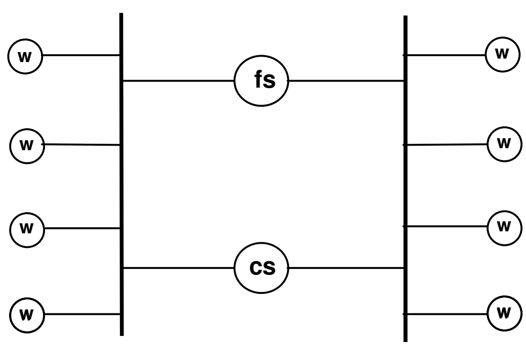
The interface you see when you log in might look very much like a MS Windows session. Don't be fooled, one of the greatest features of Unix is it's flexibility. What you see is the result of a "window manager". The default one used at CDF is named "ice" but there are numerous others available, someday you may want to experiment with some others.

Ice provides a menu to launch many programs you will use often, but much of your work at CDF will be done from an "xterm" window using an interface that is very similar to what you've probably seen in the MS-DOS window of a Windows-based PC. An xterm session or two will open for you by default. You can launch additional ones as required. In an xterm window, you are using a *line-oriented* system, sometimes called a command line interface or CLI. This means that your interaction with the computer proceeds like this: it displays a *prompt*, you type a command, it displays the output from the command, and then you see the prompt again. At the end of each line of your input, you have to press the Return or Enter key.

If the command you typed launches an additional window the prompt will not return until that window closes. Usually you will put an '&' after the command to indicate you want the child process to proceed independently, eg 'netscape&'.

Workstations and servers and terminals

As mentioned earlier CDF provides a number of workstations in Bahen for your use. These are independent machines each with its own hard disk containing the operating system files and applications, and temporary files for your login session. Most of the time you can ignore the fact that your workstation is connected to a network, but sometimes it's helpful to know about it. Here's a picture of the network connections:



In this diagram, "w" indicates a workstation.

The node marked "fs" is a file server, and "cs" is a compute server. These are central computers that manage larger hard disks containing, for example, all users' home directories. When you log in, your centrally located home directory is made available to your workstation using the local network connections, shown by the lines between nodes.

The servers also manage communications with other networks, login sessions for users connected remotely, some heavier computing jobs, and other tasks better handled by a central service.

All these computers have names, sometimes more than one name. Workstations have obscure network identifications. You will hardly ever need to use these names, however you may sometimes want to send a message of the form “Workstation blah is acting up.”.

The file server is “marvin”, “eddie” and “penguin” are compute servers, the printers are “reduce” “reuse” and “recycle” to encourage you not to overuse them, (These names suffer somewhat from cuteness, but don’t blame us: students picked them!)

A terminal display is a session that is running on a remote system and the output is being displayed where you are. Sometimes terminals are physical devices, but at CDF you will use software to obtain a session on one of the compute servers, most likely the command *ssh* eg ‘ssh eddie’. Use of a terminal session can give you access to a machine such as a compute server with more memory, faster or more processors, or software that isn’t available on your workstation, due to licensing or some other consideration.

For further technical details about the system, look in /local/doc.

Solaris vs Linux – It’s all Unix, isn’t it?

CDF has some workstations and servers running Solaris and others running Linux. You can determine the operating system (OS) running on the machine you are using by inspecting the OSTYPE variable (eg. ‘echo $OSTYPE’). For most purposes it doesn’t make any difference, the commands and programs behave the same on both platforms; but there are occasions when you will need to be aware of what flavour of Unix your program is running on. In particular if you have compiled your program on a machine running one OS, you will need to recompile if you wish to execute the binaries on another machine running a different OS.

Where is it?—your path

In the general section on Unix, we explain that Unix commands are actually files, stored on disk like any other files. (There are some exceptions for simple commands that are built into the shell, but almost all commands are disk files.) Since for the most part your assignments will involve writing your own commands it is important to know how to run them.

When you type something like

`garble blat`

in response to the shell’s prompt, the shell begins looking in a set of standard places for a file named “garble.” The first such file it finds is executed, with “blat” as an argument.

The “standard places” to look are directories like /usr/bin, /usr/ucb and /local/bin. These places are listed in an “environment variable” called PATH. (An environment variable is a named place with a value, like a string variable in a programming language.) The value of PATH is set in your **.**cshrc file. You hardly ever need to know the contents of your path variable, but you can view it with the command

`echo $PATH`

(The “$” is necessary because without it, *echo* would just display the word “PATH”.)

You might need to deal with your search path after all, if for example you *know* there’s a command “garble” kept in a file “/where/garble”, but you keep being told

`garble: Command not found.`

That probably means the directory “/where” isn’t in your search path. If you just want to execute the command once, the easiest thing is to give the command

/where/garble ...

If your current directory (the working directory) is in fact /where you could give the command

./garble ...

since ‘.’ indicates the current directory.

If you find yourself typing /where/ too often, you might want to edit your **.**cshrc file and add /where to PATH, and then log in again.

The order in which directories appear in PATH matters. For example, you might modify the version of garble found in /where, but the behaviour of **`'garble blat'`** never changes. The command

```
which garble
```

might help you find that "garble" always executes the version of *garble* found in /whose, That would be because /whose is before /where in the line defining PATH in your **.**cshrc.

One last word on this subject, there is a command " *test* " on the default path, and many students will chose to name one of their first programs "test". Make sure you know which test you are testing.

Communications

Unix was designed to encourage co-operative work. Two features crucial to this role are electronic mail between individual users, and electronic news allowing a single user to publicize items of interest to many others.

Mail

You can read your e-mail on Unix with the original *mail* command, or with *pine* or Netscape or various other alternatives such as *emacs*. You might want to try various choices before settling on the one you think is best.

Warnings

Privacy

Electronic mail may seem like real postal mail. But it does not carry the same presumption of privacy. Your mail consists of files stored at the university's expense on computers and disks owned by the university, and university employees can read it if the situation requires it.

Ordinarily we won't read your mail; we'd rather not intrude if we don't have to. But if there is reason to suspect you or your friends of an academic offence or misuse of CDF, then we may very well read your mail, and you won't be warned beforehand.

Message size

Some sites will not accept very large messages. Try to keep your e-mail messages under 1 megabyte.

News

Occasionally your instructor will want to send some information to everyone in the class, or the people running the computer will wish to tell all users about some system improvement or less happy event. If this information can be imparted in a line or two, it may appear as a "logon message" that you see after entering your password and before getting your first prompt. If it is longer, however, it will be in the online news, which you are expected to read regularly with the command *trn*, *strn*, *xrn* or *nn*. During your first news-reading session, you'll be led by the hand, but after that, when you know what you're doing, you won't be bothered.

You can also read news with *pine* or a web browser. These programs are easy to get started with, but don't offer all the useful features of the others mentioned—and if you start to read many newsgroups, you're going to want those advanced features.

Each news article or item belongs to a "newsgroup" covering a particular topic. There are many newsgroups, and you should not try to read them all. You *should*, however, read the group called "ut.cdf.announce" and the groups belonging to courses you are taking, and you might want to read "ut.cdf.general" too. Course newsgroups are simply named after the course, so CSC999's group is called "ut.cdf.csc999h", for example.

Courses taught at Erindale and Scarborough may have CDF-based newsgroups, but they are increasingly likely to use groups maintained at their home campuses. Check groups with names beginning "ut.erin" or "ut.scar".

Posting news

You can generally use whatever program you read news with to compose and post a newsgroup message. Remember, however, that *everyone* can read your news item, even if it's in a course newsgroup, so think twice before posting. A common rule is that you should wait a couple of hours before posting an article. This gives you time to reconsider or at least tone down your posting. It's very easy to offend inadvertently, and many newsgroups are full of articles provoked by misunderstandings.

Which to use?

Sometimes it's hard to know whether to use mail or news. Generally, if you flinch at the thought of typing the list of addressees, and there's a newsgroup that all of them read, you should use news rather than mail.

Because news is totally public, you may sometimes prefer mail, which is at least a little more private. But remember: the privacy of e-mail is only relative! News is recommended for items that many people might conceivably be interested in, mail for topics that are boring to most people and perhaps somewhat private, and a non-electronic channel for truly private affairs.

In the context of an academic course, you should strongly consider sending queries about assignments to your instructor by mail. Posting too much information regarding an assignment, (particularly a message containing code, algorithms, or methodology involving the solution), in a public forum can lead to severe consequences. You could expect to receive a reply by mail, but the answer to your query might also appear in the course news, since many students are likely to be interested in the answer, once the instructor has edited out any excessively revealing details.

The outside world

CDF functions independently of other systems, but does have connections to other computers and networks. CDF is on the Internet, so if you have an account at an Internet service provider at home you can access CDF from home. (In fact as a University of Toronto student, you are entitled to a "UTORdial" account giving Internet access.)

This guide will provide only brief detail on how to interact with CDF from outside, see the web pages under http://www.cdf.toronto.edu/workathome/ for more details on how to obtain software or configure for your home computer.

File transfer

You may want to transfer files between your home computer and CDF. Over the Internet, you can do this directly with an "sftp" (secure file transfer protocol) program on your PC. There are two similar commands, "scp" and "sftp". sftp provides an interactive session.

For example you can copy your files from CDF with the *scp* ("secure copy") command:

```
scp user1@machine1:file1 user2@machine2:file2
```

This copies file1 on machine1 to file2 on machine2. The machine name can be omitted if it is the one you are logged in on, as can the usernames if they are the same on both systems. You can also copy directories; read *scp*'s manual page for details.

For example, to copy the file "chaos.c" from your account "g2foobar" at CDF, to your current machine you would give the command:

```
scp g2foobar@cdf.toronto.edu:chaos.c chaos.c
```

The original file is not altered in any way. You can then work on your new copy. If you want to ship the new version back to CDF, you say :

```
scp chaos.c g2foobar@cdf.toronto.edu:
```

Notice that the file name on CDF is omitted, because it's the same as the name on the local system; we could have done that the first time. The original version of the file on CDF is now replaced with the new one.

Another possibility is simply to bring a diskette to campus and use a workstation's floppy drive. There is a set of commands allowing you to transfer files between the MS-DOS floppy disk and the Unix hard disk. Read "**`man floppy`**" for more information. On some workstations to eject your floppy you will need to use the command "eject".

Removing line feeds

One unpleasant surprise when you've transferred a file between Unix and MS-DOS is that the two operating systems treat end-of-line differently. The result is that a file that was fine on your home machine is unreadable after transfer to CDF, or vice versa.

If you are moving a file by putting your floppy into the disk drive attached to a CDF workstation, you can avoid end-of-line problems completely by using the **`-t`** option of the *mcopy* command. It's in the manual page.

You can also convert a file from one end-of-line convention to another with the *flip* command:

```
flip -u
```

changes from MS-DOS ends-of-line to Unix, and

```
flip -m
```

changes from Unix to MS-DOS.

Remote shell access

Because of licence restrictions or other factors the software used by your CDF course might not be available on your home computer. You might prefer to run your program at CDF. You can access your CDF account by ssh from another machine on the Internet. ssh (Secure SHell) is a command or protocol you can use on one Internet-connected computer to log in to another Internet-connected computer.

To connect to CDF, use the address

```
cdf.utoronto.ca
```

or

```
cdf.toronto.edu
```

(They're equivalent.)

Security is a serious concern. Other systems or unix books you have read may have mentioned similar capabilities but called it 'telnet'. Telnet into CDF has been disabled due to it's inherent insecurity, but if you must connect from CDF to some other system that unwisely still allows it, the ' *telnet* ' command is available.

Language processors

This handbook can tell you what languages are available and what commands you will use to compile or interpret them. You have to read other documentation and listen to your instructors to find out the rules of the languages themselves. Given a beginner's knowledge of a language, the manual page for the compiler or interpreter generally tells you much more than you need to get going, because the default mode typically is very simple to use and covers almost all your needs. Generally, for a compiled language like C you just say

```
gcc prog.c
```

and the compiler produces an executable file for you. For an interpreted language like LISP you say

```
cl
```

and there you are, in the interpreter. In this case it's important to read the manual entry at least to the point where it tells you how to get out of the interpreter. This is sage advice for editors also. In the C shell, you can usually suspend the job with control-Z, but there is always a better way.

So here's a list of languages available, with the name of the processor and a word or two on the kind of uses each language is typically put to.

Java:

Compiler: *javac*. Interpreter: *java*. Java is both compiled and *then* interpreted—a process you're familiar with from your first-year courses. In addition to the commands from the JDK, CDF tries to offer a more advanced development environment, ideally the same one that is used at CDF-PC, provided there is a Unix version available.

Java isn't used in many upper-level courses yet, but that may be changing.

C: Compiler: *gcc*. The **-o** option is useful, unless you like all your executable files to be called "a.out". Used for general programming; the language Unix is written in. Lets you do anything, including writing unfixably messy programs. Used in many courses, starting at the second-year level.

C++:

Compiler: *g++*. Like C, but with object-oriented facilities thrown in.

FORTRAN:

Compiler: *f77*. Mostly for numerical programming, as in CSC 336, 350, 351.

LISP:

Interpreter: *cl* (for "Common Lisp"). One of the two major languages used in artificial intelligence. Used in courses on programming languages (CSC 324) and AI (CSC 384 and some fourth-year courses).

Prolog:

Interpreter: *cprolog*. The other major AI language. Same uses as LISP.

As you can see from this list, languages *not* supported on CDF include, but are not limited to, Ada, BASIC, COBOL, Dynamo, Logo, Modula 2, PL/I, RPG and APL.

Elementary Unix

Typing

Some keyboard keys or characters have special meanings in Unix. Here is a list of some of them, giving the generic name, the actual key designated for each particular role in your initial account setup (which you can change), and a brief explanation of the purpose of the character.

Interrupt (initially control-C): The interrupt character allows you to stop a running program.

Erase (initially the "Back Space" key): The erase character removes the last character you typed on the command line, and usually does the same thing in text editors and other contexts requiring you to type responses.

Kill (control-U): erases all characters back to the beginning of the line.

End-of-file (control-D): can be used to mean "end of input" to a program, or "logout" when that is appropriate.

The allocations of keys to these roles can be changed. For example, many people like to set the interrupt character to be the "Delete" key instead of control-C. Figure out how to do this later on, if you like.

Logging off

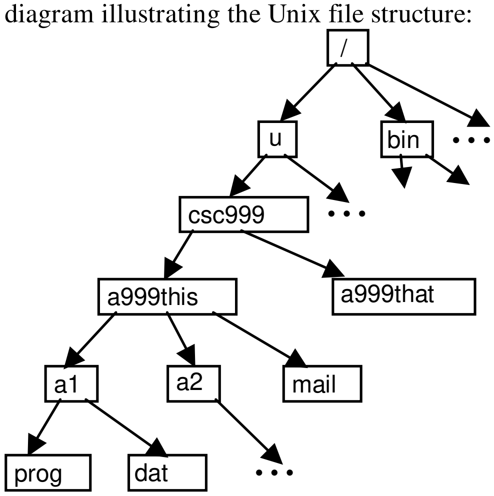
Here's a very important command:

`logout` causes you to be logged off the computer. Do it before you leave. There are numerous ways to end your login session, (switching the machine off is **not** one of them), whichever method you choose, you should see the machine return to the login prompt. You can then leave.

Files in Unix

Unix files are arranged in a tree structure. The leaves of the tree are the useful files: the data files that you might want to work on, and the programs that you might execute to do the work on the data files.

The internal nodes of the tree contain only pointers to other nodes. These internal nodes are called *directories*, and as in any tree, directories can contain pointers to other directories. Here is the traditional diagram illustrating the Unix file structure:



Here we imagine that you are enrolled in the course CSC 999F, and that your account name is "a999this". There is a directory called "a999this", just like your account, and your files are located in this directory—or rather, in the subtree rooted at this directory. For this reason, it is called your *home directory*.

According to the diagram, you own a file called "mail" and two subdirectories "a1" and "a2". Presumably you are developing a program in a1, since it contains two files called "prog" and "data".

Meanwhile, your friend's home directory is "a999that", and both these directories are children of the course directory "csc999h". Along with the directories for other courses, it is a child of the directory "u", and "u" and its other siblings are children of the root directory. In Unix, the root is always called "/"; the slash character is especially significant, as we shall see in a moment.

All the files you can use are in a single tree structure. They may be on different disks, and they may belong to different users, but they are in the same tree.

Every file can be named by specifying the full *path* from the root to that file. You specify the path by naming every directory on the path, separated by slashes and followed by the name of the file. Here are some examples:

/u/csc999h/a999this/mail –your mail file

/u/csc999h/a999this/a1/prog –your program file for assignment 1

/u/csc999h/a999this –your home directory

/ –the root directory

The name of the root is a special case, but a very natural one.

Every object in the file system is a file. That includes directories, for example: we saw that your home directory has a name, just like your program file. The same rule applies to system commands and other programs.

For example, the file-listing command "ls" is actually a file itself, probably "/usr/ucb/ls". Most of the Unix commands are kept within the directories /usr/bin, /usr/ucb, /etc, /local/bin and perhaps some others.

When you give a command, Unix looks for a file whose name is the same as the command you give. The command "ls" can be used to list all the files contained in a directory, so you could say

`/usr/ucb/ls /u/csc999h/a999this`

to list all the files in your home directory. In this example, the first file name, "/usr/ucb/ls," is used to locate an executable file or program to be run, and the second name is an argument used by the running program. Here the argument says which directory to list.

Obviously it would be a major nuisance to have to specify full pathnames all the time, and in fact you can usually list your files just by saying

`ls`

How is such an abbreviation possible?

1. *The system maintains a list of paths to search for commands that you give.* Almost certainly, "/usr/ucb" would be one of these standard paths. When you give the command "ls", it would be found on this path.

2. *You have a <u>location</u> in the file-structure tree, and commands often assume by default that you're referring to your current location.* Your location is called your "working directory" or "current directory"; when you log in, your current directory is your home directory. You can change your location with the "cd" command.

If you don't specify another directory, "ls" lists the files in the working directory. This rule and the previous one together are enough to abbreviate the file-listing command we've been using as an example.

But other abbreviating rules are also available:

3. *You can specify a file by its path relative to the working directory instead of the root.* For example, to list the files in your "a1" directory, you could say

`ls a1`

instead of

`ls /u/csc999h/a999this/a1`

assuming that /u/csc999h/a999this is your working directory.

4. *Some filenames have standard abbreviations*:

. (just a dot) is the working directory.

`..` (two dots) is the parent of the working directory.

`~a999that` is the home directory of the user whose account name follows the tilde (`~`).

`~` is your own home directory.

`$HOME` is another name for your home directory.

We're omitting some details here—specifically, the question of what part of Unix provides these abbreviations. Later, you may well want to know about that.

Some Unix commands

There are too many commands for all of them to be listed here. These are a few that you need most:

`man command`
tells you about the command. *Man* displays the standard description of the command from the Unix manual. This description is often called a "manual page".

You may not need *man* much after you're used to Unix, because most commands have a simple "default" behaviour. Often, though, a simple command can become rather complex if you want to use options. For example,

```
ls
```

lists just the names of the files in your current working directory. This is usually what you want, and so should be easy to get. But if you'd like the dates the files were last modified, ask for the optional longer listing by adding the *flag* "`l`" (the letter L):

```
ls -l
```

Options are requested in Unix by flags preceded by a hyphen or minus sign, as in this example. (The example is pronounced "ell ess minus ell".) You can ask for the listing to be sorted by time of modification instead of by name, as well as asking for the longer listing, by adding the "`t`" option:

```
ls -l -t
```

But this would get a little tedious, so you're usually allowed to combine several option flags after a single hyphen:

```
ls -lt
```

As a matter of fact, the *ls* command has a surprising number of options. Read all about it in the manual page, with "`man ls`".

`passwd` changes your password. You should do this regularly.

`cd directory`
changes your working directory.

More commands

`cp file1 file2`
makes a new copy of "file1", naming it "file2". If there's already a "file2", ordinarily it will be deleted automatically before the copy is made (though your account on CDF is probably set up to query you automatically before such removals).

You can also copy whole subtrees. See the manual page.

`mv file directory`
moves the *file* into the *directory*. If there is a file already in the directory with the same name as the new arrival, that old inhabitant will be deleted (possibly after a query).

You can move more than one file at once:

```
mv file1 file2 ... directory
```

puts all the files into the directory. And

```
mv * directory
```

moves *all* the files into your current directory into the named directory. (If the named directory is *in* the current directory, then it will be one of the files you are trying to move, and *mv* will be a little unhappy, because a directory can't be inside itself, but there will be no damage. Try it.)

The asterisk in the last example means "all files in the current directory." There are several abbreviations of this sort, and they can be used in any command, not just *mv*. See the next section, "Specifying more than one file."

You can also move whole subtrees. See the manual page.

`mv file1 file2`
renames "file1" as "file2". If there is already a "file2", it will be deleted.

`mkdir directoryname`
creates a new file that is a subdirectory of the current directory. (Directories are files too! —just as internal nodes of a tree are nodes too, even if they play a different role from the leaves.)

`rm file`
"removes" or deletes a file. It won't remove a directory, unless you use the "**`-r`**" option for recursive removal of an entire subtree—*which is obviously dangerous.*

`rmdir directoryname`
removes a directory. It won't do it unless the directory is empty. This gives me a nice comfortable feeling, so I almost always use "rmdir" instead of "rm –r".

`chmod mode file`
changes the permissions on a file, so that various kinds of people can carry out various kinds of operations. See the section "File permissions."

`cat file`
displays the file on the screen.

`more file`
does the same as *cat*, but with some screen control, so that the file doesn't zip out of view before you can read it.

`less file`
does the same as *cat* and *more*, but with even more screen control, so that you can move backward and forward in the file.

`print file`
sends a file to the printer. Don't do it, unless you truly need the paper output!

`echo words`
displays the words on the screen.

`diff fileA fileB`
compares the two files named, and reports on the differences.

`grep pattern filename`
displays all the lines from a file that contain the given pattern.

Specifying more than one file

Often you want to apply a command to a list of files, rather than just one at a time. You can just type out all the file names if you want, but there are various shortcuts to make your job easier.

`*` means "all the files in the current directory."

`*a` means "all the files in the current directory whose names end with 'a'."

`*a*` specifies files with an 'a' *somewhere* in the name.

At this point, MS-DOS users should be told that Unix cares about case: both the last two examples would include the file "Walla" but not the file "WALLA". In fact, as a Unix user you should begin to expect lower case as the default, and use upper case only when you want to shout.

If you want to match a single character instead of a string, use '?' instead of '*':

`?` means "all the files with names consisting of a single character."

`?.out` matches all file names consisting of a single character, followed by a period, followed by "out". For example, "a.out" and "2.out" will be included, but not ".out" or "prog.out". You would use "**`*.out`**" to match "prog.out"; the section *"Dot" files* on page 7 tells how to have ".out" included.

Now it's time to mention two more things.

- Blanks in files names are legal, but not a good idea, because blanks also separate the files in a list of command arguments.
- Don't call a file "*" or "?" or " " (or several other things), even though those are legal names. You'd have trouble referring to them without including other files, or else you'd have trouble even listing them.

  Names beginning with a period are also special; see the section "Dot files."

Let's conclude with a famous command that you probably should never use:

```
rm *
```

You can see why this is dangerous!

Standard input and output

Many Unix commands send their output to the "standard output," and they may take their input from the "standard input" as well. Standard input and output are often the keyboard and the monitor, but they can be attached to files, too.

For example, the *cat* command ordinarily takes a file—perhaps several files—as an argument, and copies the contents of the file to the monitor. This version will display the contents of the files "junk" and "splat" on the screen:

```
cat junk splat
```

If you want to make a new file called "both" that contains the contents of "junk" followed by the contents of "splat", use *cat* like this:

```
cat junk splat > both
```

The character '>' *redirects* the output of the command: it connects the standard output to a file instead of the monitor.

Similarly, if you wanted to, you could use '<' to connect the standard input to a file:

```
cat < junk
```

This particular example is a somewhat unlikely use of redirection, because (like many Unix commands) *cat* treats a single file argument as if it had been redirected from the standard input. But in other situations, input redirection can be very useful.

File permissions

You can't actually edit a file unless you have permission to do so. Naturally, you'd tend to expect to have permission to edit one of your own files—but that doesn't have to be true, for Unix allows you to take away that permission, from yourself or from others. (You might want to take away your own permission to protect the file from accidental editing. Of course, you can give back the permission that you took away.)

There are three kinds of permission:

read permission allows the user to view the contents of a file, use it as input to a program, or, if it is a directory, list the files that are its children.

write permission allows the user to change the contents of the file, overwrite or delete the file, or, if it is a directory, add new child files or delete old ones.

execute permission allows the user to run the file as a program. If the file is a directory, "execute" permission allows you to access its children; of course, the permissions of the child files themselves also control your access.

If you have execute permission on a file, then you can treat it exactly as you would any other Unix command. If the file is called "myprog" and was created by compiling a program that expects two arguments, you can call it by saying

myprog arg1 arg2

after the prompt. The standard Unix commands are files too: look in /usr/bin, /usr/ucb and various other directories to find them.

Now that we know the kinds of permission that exist in Unix, we can consider the kinds of users to whom they apply. For any file, there are three classes of user:

owner: the user who created the file, probably. To find out the owner, do

ls -l filename

group: other members of the owner's group. Since the owner can belong to several groups, while a file can only belong to one group, the concept of groups can be complicated, and I plan to ignore it as much as possible. Try reading the manual pages on the commands *ls*, *chmod*, *chgrp* and *groups* if you need to find out more.

others: everyone else.

So there are nine kinds of permissions for a file: read, write and execute permissions for owner, group and others. These are usually thought of in that order; for example,

rwxr-x---

would be the list of permissions for a file on which the owner has all three permissions (**rwx**), other group members have read and execute permissions but not write permission (**r-x**), and everybody else has no permissions (**---**).

The command "**ls -l**", as we already mentioned, gives a longer listing of the files in the current directory (or another directory if you specify it). This listing includes the permissions for each file at the left, with an additional tenth character to the left of the permissions that indicates whether the file is a directory (and has some other more esoteric uses as well).

To change file permissions, use this command:

chmod mode filename

Here "mode" describes how you want the permissions changed. The simplest type of "mode" states the people affected and the kind of change to be made, using characters to symbolize the meaning. For example,

chmod g+r filename

adds *read* permission for the *group*, while

chmod u-w filename

takes away *write* permission from the *user* who owns the file—that is, you yourself—perhaps to prevent accidental removal. For more details, read the manual page for *chmod*.

Editing files

Unix offers several commands allowing you to edit the contents of files:

pico is a simple PC-style editor that you can use on files containing programs, data or other text. Unix snobs may laugh at you for using it instead of some more "professional" editor, but if find it easy then you go ahead and use it while you're learning Unix. You can't learn everything at once.

nedit is a pretty easy X Window editor. That is, you can't use it in a terminal session, but it works nicely in its own separate window. We'll discuss the X Window environment later.

vi (pronounced "vee eye") is a standard Unix text editor (and you may already have seen a PC version). It takes some learning, but is quick once you're used to it. Typically, you create a program with *vi*, save it, compile it with a command such as *gcc*, and then run it.

For a helpful introduction to *vi*, see the directory

/u/cssu/pub/doc/vi

where you should start by reading the file "Readme".

emacs is the other standard text editor. Unix users tend to fall into either the *vi* camp or the *emacs* camp, but there's no reason why you shouldn't try both.

Emacs itself provides a tutorial. Read the first screen it shows you to find out how to proceed.

ed is a line editor. It was the original Unix editor, though the local version has been modified to make it a little friendlier. For instance, it has prompts, and help is available.

Probably you won't learn it, but you'd be surprised how much easier it is to do global changes (throughout the whole of a file) using *ed* instead of any of the full-screen editors. And screen editors like *vi* incorporate features descended from *ed*, so knowing *ed* can be a real help.

"Dot" files

Earlier we mentioned the command '*startupfiles*' which restores the default system files. If a file name begins with a period, the *ls* command won't display its name unless you use the "**-a**" (for "all") option. Examples we've already seen are "**.**" and "**..**": every directory includes entries for both itself and its parent, but *ls* doesn't list them without the **-a** flag. Try it!

There are lots of files you usually don't want to be bothered with, including configuration files that determine how your account will behave and command files that are run by specific programs when they start up. Often their names will begin with a period to make them invisible, and commonly they end with "rc" (for "run command"). Some examples:

.login a file describing how your account is to be set up when you log in.

.logout a cleanup file, executed when you log out.

.cshrc used by the "shell" when it starts up. The next section discusses the shell.

.mailrc used by the *mail* program.

.newsrc used by many news programs.

Your shell

You don't need to know *very* much about how you interact with the operating system, but it does help to know a little.

The basic fact to understand is that *at all times* a program is running. That's right: all the time, even when you're just sitting there thinking about what to type after the prompt. You may think that at these times you're dealing directly with the operating system, which you've heard manages interactions with users as well as allocating time to various tasks, looking after input and output, maintaining the file system, and so on.

But that's not true. You deal not with the operating system itself but with a *shell*, a program that mediates between you and the operating system. When you ask for a listing of your files, your "ls" command is collected by the shell and interpreted to find out that you want to run the "/usr/ucb/ls" program. Then the shell asks the operating system to start up "/usr/ucb/ls", which lists your files and then asks the operating system to start the shell going again.

This may sound cumbersome, but in fact it's very handy. We want the operating system to concentrate on its own major concerns without fussing about making things look good to users. That leaves the shell to pay careful attention to things like editing the command line when you type a back space, remembering your working directory, and locating programs you want to execute.

What happens is that the shell program is started up (by the operating system, of course) as soon as the logging-in process is finished. When it starts up, it reads the file "**.**cshrc", containing customizing commands. Consequently, you can modify the behaviour of the shell by changing the contents of **.**cshrc. For example, you can change the characters used for interrupt and backspace, you can set up aliases—short commands that you use in place of long, tedious ones—and so on.

Every time you open a new window, a separate copy of your shell is set up to control it, and it begins by executing the commands in your **.**cshrc file. (This is different from your **.**login file, which is run only once, when you log in.)

There's a lot you can put in your **.**cshrc, too much to cover here. The default shell we have chosen to set up your CDF account with is *csh* or the "C shell," after the programming language C. In fact, rather than the standard C shell *csh*, we've given you a variant called *tcsh*, but the differences are minor.

There do exist other shells. The traditional alternative is *sh*, the "Bourne shell". *Sh* has its uses, but it is less convenient for stopping and restarting jobs ("job control") and reusing previous commands ("history"), Just like other programs, the various shells have manual pages. Try "man csh", "man sh", or "man tcsh" to find out more, probably, than you can easily absorb about your shell.

Shell scripts

If you find yourself typing the same set of commands over and over, perhaps with a few simple variations, you can save yourself work by writing a "shell script". A shell script is a file containing a sequence of shell commands, just like the ones typed at the keyboard. (The MS-DOS equivalent is a "batch file".) The commands in a shell script can include loops and conditionals, and they can refer to the command-line arguments.

Almost all Unix programmers prefer to use the Bourne shell, *sh*, for their shell scripts, rather than the C shell.

X Window

Up to this point we've covered CDF from a command line perspective. Windowing systems are standard on personal computers, so presumably you are already familiar with something like what CDF offers. However, though such systems share a common heritage, there are differences in detail.

The particular windowing system used at CDF is "X Window", sometimes called just "X". Other windowing systems are available for Unix, but X is the one used here.

Windows

You'll find your workstation screen contains numerous windows. You are logged in to the workstation separately in each of these windows. If you type "who" in any of the xterm windows, you'll see yourself listed once for each window.

Notice that all the windows have a special bar at the top, called the *title bar*. At the left of the title bar is a little square containing a symbol that looks a bit like a hyphen, "–". This is called the *window menu button*. The middle of the title bar contains the name of the window. The use of the buttons will be explained later, but you are probably familiar with the basics from similar systems.

Mice

The mouse on a Unix workstation has three buttons, and all of them have a role to play (in contrast to PCs, where the middle mouse button really doesn't do much).

In X Window, to work in a particular window you simply position the mouse pointer in that window. You don't have to click any of the mouse buttons, but you do have to keep the pointer in the appropriate window. This is different from MS Windows, and it may take you a while to get used to it. When the mouse pointer is in a window, the background to the window's title bar and the window's borders change appearance to indicate the window has the focus.

Using the mouse

Here are some useful mouse operations:

Click in a window's title bar: Even if a window is currently active, because the mouse pointer is on it, it may still be partly covered by other windows. To "raise" the window so that it is completely visible, click in the title bar with the left-hand mouse button. To "lower" it—put it under the other windows—click on the title bar with the middle mouse button.

Click and hold: Holding a mouse button down often displays a pop-up menu.

Menus

In X Window, menus can appear anywhere on the screen, and the particular menu you see will depend on the context: where you are, which mouse button is pressed, which keyboard keys are pressed, and so on. Very often it is the right mouse button that gets you a menu, while the other buttons do nothing.

Unlike MS Windows, X Window requires you to hold the mouse button down while moving to the menu selection you want.

Sometimes you may need to press a key on the keyboard at the same time as you press a mouse button. For example, as discussed below, you can customize a terminal window by holding the "Control" key down while pressing a mouse button.

On the background (sometimes called the "root window"), holding down the right mouse button brings up a menu of interesting choices.

In the title bar, holding down the right button, or clicking on the window menu button in the top left corner, gets you a menu giving various options for changing the window's appearance, such as resizing, moving, raising and lowering. Some options, like resizing and moving, start an operation that continues until you click a mouse button.

Within a xterm window, when the Control key is pressed on the keyboard, all three mouse buttons give you menus affecting the appearance of the "terminal." For example, you can obtain a scroll bar that allows you to look back through previous output to the screen, using the mouse buttons to click and drag in the scroll bar; you can change the size of the type in the window, and so on.

Warning: With the Control key *not* pressed, the left and right mouse buttons cause text in a terminal window to be selected and stored in a buffer. This is all right, but if you then press the *middle* mouse button, the saved text is sent to the terminal session as if you'd typed it as a command. This is hardly ever appropriate unless you want to cut and paste in a text editor. When you mean to issue commands to the shell, it can be disastrous. Be careful with the middle mouse button!

Screen buttons

A screen *button* (not to be confused with the mouse buttons) is a small part of a window that you can click in to cause some frequently needed action. The behaviour of windows with the ice window manager is probably very similar to what you are used to.

The minimize button at the right end of a window's title bar shrinks the window. The window still exists, and if it is a terminal session, still contains the output from any running program. This is like the "minimize" button in MS Windows. The maximize button, next to the minimize button, allows you to increase the size of the window. You can also use the "maximize" option from the right-hand mouse button's title-bar menu.

Programming X Window

The X Window system has many more capabilities than we have space for here. For example, you can open and close windows and direct output appropriately from within programs. The Internet and a good bookstore are useful resources in learning about these possibilities.