

A Student's Guide to CDF

[Welcome to CDF](#)

[This Guide and Other Sources of Help](#)

[Where are the PC's?](#)

[Rules](#)

[Your Account](#)

[Limits and Quotas](#)

[Finding your way around the CDF system](#)

[The Outside World](#)

[Elementary Unix](#)

[Good Citizenship](#)

Welcome to CDF

Computer Science courses offered on the St. George campus 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 supporting computing education.

Keeping informed about Computer Science Undergraduate matters.

The department displays information about undergraduate matters in a number of ways. It would be useful to you to bookmark the [Computer Science Community bulletin boards](#) page, as it displays a calendar with important academic dates, plus events, sessions with companies and job information. Announcements are also made to the news group [ut.cdf.announce](#), and as you login to the cdf computers on campus, remember to read the message of the day.

[top](#)

This Guide, and other sources of help.

How to read this guide

This guide assumes you, the reader, are a student in a Computer Science course using CDF. While primarily intended to help those going from first to second year courses at CDF, even students who have been here a while might be reminded of useful things by glancing through this handbook.

You will be expected to use the command line more often after first year, but you should find what you learn't from [First Year or New Users](#) guide useful. A small amount of the information in the first year guide is repeated (for example the location of labs).

Because of this background, this guide covers a collection of important information about

what you can do at CDF, what your responsibilities are, and how to discover more details. It is also an introduction to Unix for people unfamiliar with using the command line.

Other sources of help

This handbook can only give brief advice about Unix 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, they are 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 [CDF FAQ](#) (or type *faq* on the command line) and the [CDF home page](#).

[top](#)

Where are the PCs?

Gerstein Science Information Centre

[\(at the Sigmund Samuel Library, 9 King's College Circle\).](#)

Open during library hours, this room is at the north end of the second floor. To get to it, go into the main entrance of the library, and pass through the turnstiles. To the right of the Loan Services desk, there is an elevator and stairs. Go to the second floor, and the computers are in room 2360.

Bahen Centre for Information Technology

[\(.40 St George St.\)](#)

Contains a number of labs: BA2210, BA2220, BA2240, BA3175, BA3185, BA3195 and the Great Hall with between 12 to 24 machines each. 24 hour access with student card, except during tutorials. BA3175 and BA3195 are used for tutorial sessions during the day (M-F). They are available for general student use evenings and weekends, and at times during the day when they are not booked.

[top](#)

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 there are some basic rules governing your behaviour.

University Rules

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](#).

Rules for CDF users

Each time you login, you are agreeing to the rules of conduct in the labs. You can read the rules [here](#).

You are sharing these resources with other students, so please read this thoroughly as you will be held accountable.

Academic offences.

Sometimes 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.

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 threaten system security or interfere with other users may require us to suspend your account immediately. Sometimes it may be a little unclear *which* set of rules is relevant, but it is very clear that *some* set of rules 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.

[top](#)

Your account

Course accounts

Each student using the system must be enrolled in a Computer Science course. You will automatically be issued an account, and email will be sent to you if you have a valid email address in your ROSI profile. To use it, you will need to know your account name (also called your "logon name", "login"; or "user name") and your password. You will use the same account for all your Computer Science courses, unless you are accepted into a CSC program, in which case you are eligible to request a program account. Accounts are prefixed by "c#", where # corresponds to the last digit of the year you get your account (eg 5 for 2005) and then the first six letters of your last name, as it appears on your student card. If the first six characters are shared by more than one person, accounts created later will have the last character shifted farther down in the alphabet: clarke, clarkf,... clarkh. If your last name is shorter than 6 letters, letters from your first name on your student card will be used to fill.

<i>Name</i>	<i>Account/Login</i>
CLARKE JAMES	c5clarke
CLARKE THOMAS	c5clarkf
LI JENNIFER	c5lijenn

When your account is created, you will be sent email (to the address you supplied in ROSI) with the details. If you forget your account name, please use [CDF Username Lookup](#). Your password will initially be your student number, but you must [change](#) it the first time you log in. 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. 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 may be deleted.

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 ask the system administrators to suspend your account while you are away. 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. Program accounts are set up at the start of the next term after you have requested the account. You can request a [program account online](#) .

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.

Changing your password

The first time you log in to CDF, you must change your password.

You will be asked to read the CDF Rules, and then to change your password. You have to enter your old password, and then the new one *twice*, to make sure you're typing it consistently. Your new password will only be accepted if it passes a program designed to securely check passwords. If it does not, you will be asked to enter another one. Good password rules require a mixture of upper- and lower-case letters (from the usual European alphabet), numbers and punctuation (but no spaces). It must be between 7-8 characters in length. The idea is to make it harder for an attacker to guess your password: "hiMom!" or "TrustNo1" are harder to generate automatically than "mymother" and "TRUSTME". On the other hand, you need to be able to remember your password. Pick something that's not totally random, such as a phone number combined with a name ("978-Fido") or a derivative of a memorable phrase (eg. Wylfw? ~W Would you like fries with that?).

Choose your password before going to CDF to change it, so you'll have a clearer idea of whether you really can remember it.

Change your password during the term.

The easiest way to change your password is to type

passwd

at the command line. You will also find Reset Password in the menu.

Forgot your password?

We can't tell you what it was.

Show your photo ID card to a system administrator in their office in BA3224 to have it reset to your student number. You must then reset it to what you want.

Logout

It is essential you logout before leaving a computer. If you do not, and someone copies your work, you are still responsible.

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.

[top](#)

Limits and quotas

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

Disk Quota

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 20 megabytes per course, and is increased if you do more courses. 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 other media for backup.

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.

You also have a disk quota for your mail INBOX. The quota is 2 megabytes, with a hard limit of 5 megabytes. If you go over the 5 megabyte quota, mail can no longer be delivered. Remember to move your received mail from the INBOX to another folder, as the other mail folders are stored in your home directory, whereas the INBOX is stored on a separate system (with the low quota).

The command *quota -v* will tell you how much disk space you're using. It reports your usage of your home directory and your mail INBOX. 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 programming course, and less for a theory course. 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, in increments of 300. If you are doing more than one course, you can call up more of your print quota with the command *pquota -i*. 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 BA3224 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. This means a student cannot lock the workstation to reserve it.

[top](#)

Finding your way around the CDF system

[Interaction with the system](#)

[Restoring startup files](#)

[Restoring KDE default startup files](#)

[Where is it??your path](#)

[Deleting files](#)

[Printing](#)

[Security](#)

[Workstations, servers and X terminals](#)

[Communications](#)

Interaction with the system

To begin using a CDF workstation, you first log in, providing your account name and password in the usual way. Remember to [change your password](#) 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 its flexibility. What you see is the result of a "window manager". The default one used at CDF is named "KDE" but there are numerous others available, someday you may want to experiment with some others. KDE 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.

The KDE "xterm" can be launched from the console icon on the panel or by typing *konsole* on the command line. If you want to launch a traditional xterm, type *xterm* at the command line. 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 'firefox&'.

[top of section.](#)

Restoring startup files

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 default startupfiles.

[top of section](#)

Restoring KDE default startup files

The default window manager is KDE. You are able to customize your desktop in many different ways.

If you make changes to the appearance of your desktop that you do not like, remove parts of the menu or if you manage to make your KDE unusable, you can run

```
/local/bin/kdedefaults
```

to restore the KDE defaults. This will take effect the next time you login.

[top of section](#)

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 will execute the first file named "garble" it finds, with the argument "blat".

Where does it find the command?

You have an "environment variable" called PATH that lists where the shell must look, and the order in which it must look. (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.

Note. This is when you might end up with the wrong order in your path, such that you can no longer run the commands that were working before. Read the [Restoring Startup Files](#) again. 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.

[top of section](#)

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 guaranteed 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.

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.

[top of section](#)

Printing

For most of your printing needs, you will be able to print directly from the software you are using. The printing will be printed to the closest printer, unless you decide to change this. The printers are as follows:

<i>Location</i>	<i>Closest Printer</i>
2nd Floor, Bahen	p2210a and p2210b in BA2210
3rd Floor, Bahen	p3185a in BA3185
Gerstein Lab	gerstein in room 2360, the Gerstein lab

Whenever you print something, the pages are counted and your printquota is reduced accordingly.

Where are the printers?

The Printers are located in the following rooms:

p2210a (BA 2210)	p2210b (BA 2210)	p3185a (BA 3185)	gerstein (Gerstein 2360)
-------------------------	-------------------------	-------------------------	---------------------------------

If you run the Command 'printers' it will display this information.

How do I print to the Printers?

You can print to the printers in the following ways:

1: Use the “Print” command from the Graphical menu in most programs such as Firefox (File – Print)

This method of printing usually works. To find out which printer is the default for your workstation, select the PRINTERS icon on the KDE menu. Alternatively, in a terminal window type 'echo \$PRINTER'

2: Using the command line to print postscript files (*filename.ps*)

a: Print to a file: Most programs allow you to send a print job to a file. Save the file as '*filename.ps*' This will also allow you to view your print job using the program 'gv'.

b: Use lpr or print command to print your file. The syntax is:

<code>lpr <i>filename.ps</i></code>	<code>print <i>filename.ps</i></code>
prints a postscript job (.ps) in single sided	print sends a job to the printer double sided

To direct your job to a specific printer, use the -P option in the following way

```
lpr -Pprintername filename.ps
```

3: Printing PDF's:

a: Using Acroread: start the program 'acroread' from the menu or typed into a terminal.

Print your PDF document using the “File – Print” method. This is the recommended way to print PDF documents to ensure the most consistency.

b: Using the command line:

The 'print' command will convert your pdf to ps and send it to your designated printer in double sided pages (see above). The command 'lpr' will NOT print a pdf document. It must first be converted to postscript (ps) using the 'pdf2ps' command. 'pdf2ps *filename.pdf*' will create an equivalent *filename.ps*

How do I see the status of my print job? How can I cancel it?

Each job has a "job number". You can find the job number of all the jobs currently printing or waiting to be printed using lpq which stands for Line Printer Queue. (Unix usually calls its printers "line printers" for historical reasons.) Like most printer commands, lpq can take a -P argument; use -P*p2201a* to see the queue for the first downstairs printer, and -P*p3185a* for the upstairs printer. The output will look something like this:

Rank Owner	Pr Job Host	Files	Size	Date
active wayne	Z319b2210-99	(stdin)	7	Jan 6 23:51
2nd you	Z 320 frood	foo.txt	7	Jan 6 23:51

meaning that wayne's job (number 319) is currently printing, and job number 320, belonging to "you", is next. Both jobs are 7 bytes. To remove your job, type `lprm 320`. Don't forget to use `-Pprintername`. You can sometimes remove an active job, but often the printer will have printed it before the command is processed, unless it has jammed or is otherwise delayed.

[top of section](#)

Security

You have already read the [rules](#). You must be aware of general security.

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, and there is a brief description about [File Permissions here](#). 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](#).

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". Read more about the [command ls](#), or `man ls`, to find out how to read the information about your files.

[top of section](#)

Workstations, servers and X terminals

As mentioned earlier CDF provides a number of **workstations** in Bahen and Gerstein 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 you will notice that wherever you login, your home directory follows you.

There is a central **server** that manages larger hard disks containing all users' home directories. When you log in, your centrally located home directory is made available to your workstation using the local network connections.

Other central **servers** 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 servers have names, sometimes more than one name. Workstations also have names. and you can see the name of the workstation you are using when you login. 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."

At CDF you can also use software to obtain a session on one of the compute servers, most likely the command *ssh* eg 'ssh cdf'. 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.

[top of section](#)

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 *thunderbird*, a Gui interface, or with the original *mail* command, or with *pine* 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 on-line community boards, which you are expected to read regularly.

[top of section](#)

[top](#)

The outside world

CDF functions independently of other systems, but does have connections to other computers and networks. Because of licence restrictions, special software used by some CDF courses may not be available at other sites. This guide will provide only brief detail on how to interact with CDF from outside,

Remote Access using the NX Server

CDF offers an NX server for remote desktop connection from home or from your laptop. The NX technology attempts to greatly improve the experience of remote graphical connection over a slow Internet link by optimising perceived speed of response and allowing one to suspend a graphical session and reconnect to it later.

In order to take advantage of this method of connection to the CDF, you need to download, install, and configure an NX client software, which is available free of charge for Windows, MacOS, and Linux operating systems.

To access these instructions, you need to have an active CDF account. If you remember your CDF user name and password, proceed to the [instructions](#).

Remote Access using SSH

You can access your CDF account by ssh from another machine on the Internet. ssh is a command or protocol you can use on one Internet-connected computer to log in to another Internet-connected computer. CDF is on the Internet, so if you have an account at an Internet

service provider at home you can log in to CDF with *ssh*. Assuming your account name was foobar, you would

To connect to CDF, use

```
ssh foobar@cdf.utoronto.ca
```

or

```
ssh foobar@cdf.toronto.edu
```

(They're equivalent.)

This will give you a command line session with the CDF system. The CDF system actually consists of multiple compute servers, and by accessing them using the name *cdf*, the load is spread.

Very occasionally during the year, one of the compute servers might be overloaded. You would notice this if *cdf* did not respond as it usually does. Under that circumstance, you could try going directly to one of the compute servers.

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 its inherent insecurity, but if you must connect from CDF to some other system that unwisely still allows it, the '*telnet*' command is available.

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. From an *ssh* shell in unix there are two 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 CDF, where we suppose you are logged in, to the account foobar on the Scarborough computer "bluffs.scar", you would give the command:

```
scp chaos.c foobar@bluffs.scar:chaos.c
```

The original file is not altered in any way. You can then log in to bluffs.scar and work on the new copy there. If you want to ship the new version back to CDF, you say (on bluffs.scar):

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

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

The easiest way to transfer files is to simply bring a "usb key" to campus and use a

workstation's usb port.

To use SSH from a windows machine you need to use an SSH client. Openssh lists a number of clients [HERE](#).

At CDF we have had the best experience using [Putty](#).

Alternative proprietary programs are available for free download, such as [Tunnelier](#), which includes a file transfer client.

[Filezilla](#) is useful for transferring files between your cdf account and your windows PC using a graphical interface.

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 usb 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.

[top](#)

Elementary Unix

[Typing Files in Unix](#)

[Some Unix commands](#)

[Specifying more than one file](#)

[Other Important Matters](#)

[Standard input and output](#)

[File permissions](#)

[Editing files](#)

["Dot" files](#)

[Your shell](#)

[Shell scripts](#)

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.

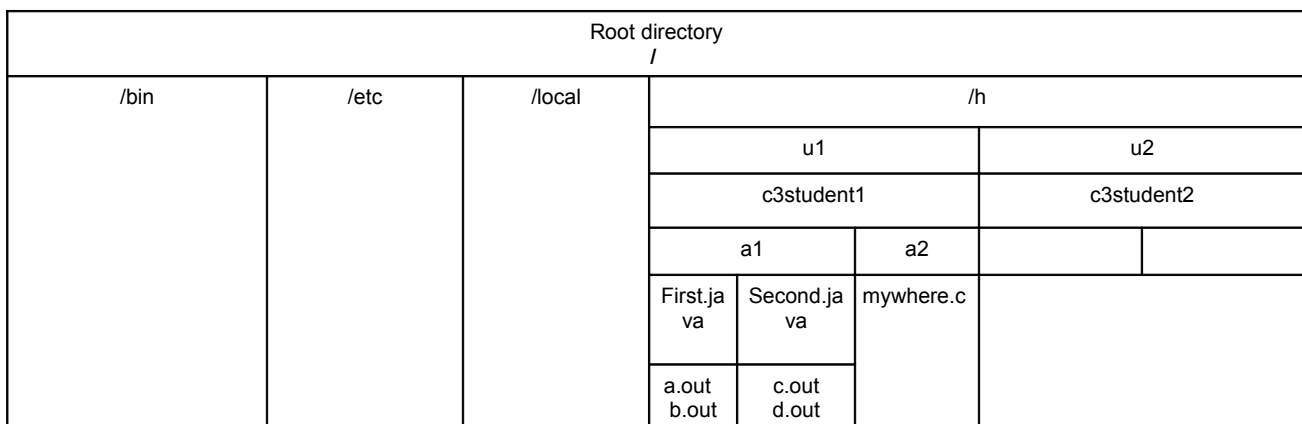
[top of section](#)

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.

The drawing below is a representation of this structure, always drawn with the root at the top.



Here we imagine that you have a course account and your account name is c3student1. Your "Home" directory is actually the subtree rooted at /h/u1/c3student1. You appear to have two assignments, and in one of them you have two files. Thus, the "leaves" in your tree are

First.java and Second.java.

Meanwhile, your friend's home directory is "c3student2", and both these directories are children of the directory "u2". Along with other directories, it is a child of the directory "h", and "h" 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:

/h/u1/c3student1/a1/First.java	your first java file
/h/u1/c3student1/a2/mywhere.c	your c assignment file
/h/u1/c3student1	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 "/bin/ls". Most of the Unix commands are kept within the directories /bin, /usr/bin, /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

```
/bin/ls /h/u1/c3student1
```

to list all the files in your home directory. In this example, the first file name, "/bin/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, "/bin" would be one of these standard paths. When you give the command "ls", it would be found on this path.

2. *You have a location 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 /h/u1/c3student1/a1
```

assuming that /h/u1/c3student1 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.

~c3student2 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.

[top of section](#)

Some Unix commands

There are too many commands for all of them to be listed here. You will be taught the use of some commands in some of your second year courses, so that is another source of information.

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.

```
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.

```
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.

[top of section](#)

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".

[top of section](#)

Other Important Matters

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.

If you manage to edit these incorrectly, go to [Restoring startup files](#).

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

```
rm *
```

You can see why this is dangerous!

[top of section](#)

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.

[top of section](#)

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, /bin, /local/bin 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*.

[top of section](#)

Editing files

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

nano

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.

[top of section](#)

"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.

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.

[top of section](#)

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 "/bin/ls" program. Then the shell asks the operating system to start up "/bin/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.

[top of section](#)

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.

[top of section](#)

[top](#)

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.

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, please 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 on the website. 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.

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 *rsc* and *cvs* for ways to keep track of previous versions of your program.

Read email without spending too long at it. This may seem irrelevant to sharing and security, but awareness of what's going on will help you cooperate with other users.

[top](#)