A Short Introduction
to
HUJI-CSE Computing Systems

system@cs.huji.ac.il

5771

Authors: Alex Kremer, Tomas Winkler, Elisheva Alexander, Rafael Horowitz, and Ely Levy.

Maintainers: Yair Yarom, Ely Levy.

This manual can be found at http://wiki.cs.huji.ac.il/ –> Friday With the System.[1]

Contents

Chapter 1

Welcome to CSE

Welcome. **Don't panic**. It will take time to become familiar with all that is said in these introductory classes. Their intention is to get you *fast* to the level where you can start doing your exercises, and give you a solid basis that you can then expand on your own.

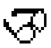
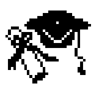
For those of you that know about computers: Please bear with us, since the (computer) world is in the details. You can miss something that could save you days or weeks of work. Most of our computing systems are new even to people with years of hacking experience.

1.1 General Information

1.1.1 About this Guide

This guide was written specifically for you! We hope you will find it useful throughout your studies. If you need a new copy you can obtain one on our web site http://wiki.cs.huji.ac.il/ on the "Friday With the System" section.

Different symbols used:

-  Warning, this section is a bit harder than the rest, you may need to reread it.
-  This section does not include practical information - it is only general information and for further reading. It is intended for nerds.
-  This section is aimed mainly at second year students. You will probably not need this in your first year.

We highly recommend that you refer back to this guide later in the year, either when you encounter problems, and/or once you are a bit more used to CSE network.

We hope you enjoy the guide, please send us any comments or questions.

1.1.2 Getting help

This guide gives general information about the computers in the School of Computer Science and Engineering, and a short introduction to linux. The most important piece of information you need to remember from this guide, is where to look for more information:

- Our wiki page: http://wiki.cs.huji.ac.il/ contains the most up-to-date information about our system.
- This guide, can be found on our wiki
- www.google.com (or yahoo, or bing, or any other search engine near you). The internet contains a lot of information and tutorials about computer in general about linux, and about specific software you will use. Though our system is somewhat customized, most of the software is standard and any tutorial you will find will probably suffice.

1.1.3 Who & Where Are We?

We are the **System**, and you will eventually be assimilated into the wonderful world of computers. The System Group is in charge of both hardware (computers) and software in the department.

From time to time we may send you various announcements by e-mail, however:

- We will never ask you for your password. **Do not** send us your password.
- We will never ask you to run any programs. **Do not** run any such requests.
- We will never send attachment for you to run.
- Exceptions:
  - When you encounter a problem, the solution might be to run some program. **However**, this program would be installed in the system, and won't be sent by mail or under some user's home directory (i.e. under /usr/.. not under ~.../ or /cs/stud/...).

We can be reached by:

- E-mail: `system@cs` - this is the best way to ask for help. We will try to reply as soon as possible (ASAP). Please, do not use our personal logins for system related issues (except if asked to do so).
  If you send your problem through E-mail to system@cs, it will be handled in a more efficient way:
  - it will get to the most qualified person on the subject
  - once solved it might help other people and us as well
  - it will not be forgotten
- Telephone: `02/(65)85691/2/3` - for urgent problems **only**, everything else should be sent by mail!
  Please call us if (and only if):
  - You cannot send mail to us
  - The building is on fire
- Do not come to the System room uninvited. For any problem first try by mail.

When to contact us?

- You have encountered hardware problems, i.e. some parts of the equipment seem broken or Dis-functional (e.g. the screen is blank, mouse is not responding, etc.)

- Software Problems, i.e. you are not able to copy your files to external media; When you try to use a certain program you are experiencing strange problems, etc. Note that although we installed most of the software we are not expert in all of them, if a course TA requires you to use a software, you should ask him what is the proper way to use it.
- General questions about our systems, **not related to your exercises**.

How to report problems?

In your mail, include any information that might help us solve the problem, without having to ask you more questions.

For example:

- Name of the computer(s) in which the problems occurs
- The full command which you ran, including the parameters
- When did this problem start?
- We would like to recreate the problem, what steps should we take?
- If there is an error message, please include it in your mail
- Be specific, do not write “the keyboard is not working”, please write which key is broken
- Does the problem happen elsewhere, on other computers?
- Does the problem happen to other users as well?
- We **do not** need your password, Israeli ID number, or any personal information.
- Please send your mail from the cs account (if you have one), or from the mail.huji account only. **Not from your private gmail or ISP account.** Whoever will send from his private account, will risk being ignored!
- From the shell, you can run the command **system-support** which will bring up a from you can fill that will be sent to us.

Please do not be afraid to contact us, if no one reports problems, they will never be fixed!

1.1.4 Computer Labs Rules and Regulations

Please note the following rules while you’re in the labs. Whoever break these rules, will have his account closed (This is **NOT** an empty threat!):

- Do not drink or eat in the computers lab
- Do not turn off any of the equipment (computers, screens, printers)
- Do not unplug any equipment (computers, screens, etc.) in the computers lab
- Do not touch the printers
- Do not connect any electrical device into the electric sockets where the computers are connected. There are designated sockets in each lab clearly marked where you can connect you devices.

Logging On

logging on is a process in which you tell (and prove) to a machine (computer) who you are. After that, the machine starts a "session" during which you work on it. After you finish you are **requested to log-off (logout)**.

➥ **Do Not** leave your login open, someone can abuse it!

➥ **Do Not** shutdown the computer after finished working on it!

1.1.5 What is UNIX, Linux, FreeBSD?

UNIX is a class (family) of **Operating Systems** first developed in the early 1970's at Bell Labs by Ken Thompson and Dennis Ritchie. It has been improving since then.

Unix is a **multi-user**, **multi-tasking**, **multi-threading** operating system. Multi-tasking means that it can run multiple tasks (programs/applications) at the same time. A multi-user operating system is one which can support multiple people using the same system at the *same time*. Each user can connect to the system (on-remote) and run their own programs without interfering with other users. A single UNIX system can support hundreds of users not just accessing its files, but also running applications simultaneously.

UNIX systems are commonly used as servers for network services such as electronic mail, World Wide Web, file and printer sharing, and more. The TCP/IP protocol, one which the Internet runs, was originally developed at the University of California at Berkeley for BSD Unix systems.

The methodology and logic of UNIX have, over the years, become paradigms behind many other operating systems. Its use of open standards and technologies makes it the tool of choice for academic purposes.

UNIX is not a single operating system. Rather, it is a family of operating systems based on a common ancestor, the original Bell Labs Unix. There are many implementations of UNIX today. Most are "source level compatible" - application written for one can easily be transferred (ported) to another. There are both commercial variants of Unix (e.g. Sun's Solaris) and non-commercial variants such as Linux and FreeBSD, NetBSD and OpenBSD.

GNU/Linux is a Free Software [1] **implementation** of UNIX. It was originally created by Linus Torvalds, who released it under the General Public License, for anyone to change and improve. Literary tens of thousands of developers are now working on it.
Its open source approach enables students to see the actual implementation of all the system components.
There are a number of organizations who distribute Linux. For example: RedHat, Debian, Libranet, SuSe, Mandrake, and Slackware are a few.

[1] Free Software and Open Source are confusing terms. You may have heard of the term "Shareware". Shareware means you may share the software among your friends. Free Software gives you much more! The word "free" in Free Software comes from the word "freedom". Not only can you share this software among your friends, you are also free to change and improve it, since the "source code" is released together with the software. The term "Open Source Software" (OSS) is another name for Free Software, however, it emphasizes the *technical aspect* that the source code is open for all to see, while "Free Software" emphasizes the *ideological aspect* of programming freedom. Most free software is legally protected under the umbrella of GPL (The GNU General Public License). GPL guarantees, that no one who wants to use such software in commercial and/or non-commercial products, cannot do so without releasing the product sources as well.

FreeBSD is another Open Source[2] version of UNIX. Most of our server machines (computers which provide services to other computers or/and people) are FreeBSD based.

Solaris the SUN's version of UNIX. Very good for network applications. Most major web sites (cnn.com, amazon.com, ...) use it.

Learning Unix is like learning a new language, and requires patience and practice to become fluent. It is a very powerful tool, and can help you work very efficiently once you learn how to use its more advanced features. It is important to remember that it is not a home user's operating system. It is the preferred power-tool of scientists and engineers.

1.2 System Overview

In this chapter we will present a brief overview of the system, later we will expand on each topic with deeper explanations. You are not expected to understand everything, you will learn how to apply all these theoretical issues in the following chapters.

1.2.1 Terms: Login, Password, One Time Password (O.T.P.)

Login: (or **username**) is your unique identifier (name) in all of our computer systems. It contains letters/digits and is the way you are known in the computer system. Some of you will soon find out that people, too, like to use logins instead of your names.

- This login is used only[3] with the CSE computer network, it's a separate account from any other computer accounts you might have in the university.
- Your entire working environment (files, desktop settings, etc.) will follow you wherever you go and use your login to start a session.
- Your **e-mail address** is also based on your login (see section 1.2.7 on page 13):
  - `<login>` when working on one of our systems,
  - `<login>@cs.huji.ac.il` from the outside.
- Your **web (www) home page** Address (URL) will also be based on your login: `http://www.cs.huji.ac.il/~<login>`
- Your **home directory** (storage space where you can work), will also be accessible through either of these two: `~<login>` or `/cs/stud/<login>`.

How to choose a login:

Take care when choosing a login name, this login will follow you throughout your studies at the Hebrew university. If you ever publish a paper, communicate with researchers abroad you will **not** want a login that looks like this:

- username@cs.huji.ac.il

[2] Open Source Software (OSS) means that all of its "source code" is released together with the program. Everyone can change, improve and use it without a fee, as long as [s]he releases the changes for all to see. Most open source software is legally protected under the umbrella of GPL (The General Public License). GPL guarantees, that no one who wants to use such software in commercial and/or non-commercial products, cannot do so without releasing the product sources as well.

[3] If you've already opened an account in the public computers (shprinzak), you'll be given the same username for your account at CSE. But for all other purposes, it's a separate login.

- idiot@cs.huji.ac.il
- slkdfjlskdjf@cs.huji.ac.il
- cool_hacker@cs.huji.ac.il

A good login name should be pronounceable, and reassembling your name.

Password: is the way you are "authenticated" to our computers, i.e. your real identity is assured. Only you are supposed to know it, thus no one else can impersonate you - it should be your best kept secret. Once "authenticated" the computer system will let you do whatever you are allowed to (e.g. submit an exercise). There are several password you will use in the Hebrew University, in the computer science department there are the unix password and the OTP. Other facilities of the university uses different passwords, such as your 4 digit code, the mail.huji.ac.il password and more.

UNIX password

In practice it is a 6-8 letter/digit/special-sign string, which should be hard for someone else to guess

- It **should not** be based on a well known fact about you (your phone number, your address, date of birth, boyfriend).
- It, also, **should not** be composed of a dictionary word (English, Hebrew, ...).
- You are **NOT ALLOWED** to pass your password to **ANYONE**, including your family members, system group members or any other person that asks you.
- We will never, we repeat, **never** ask you for your password. Do not give it to us even if we ask for it.

OTPs (One Time Passwords) are generated by a special program on your cell phones[4]. Each time they produce a new password (a string of 6-8 hexadecimal digits [0-9A-F]). If you phone is lost, or stolen, please report, so we may disable its OTP program and register your new phone (if you have one).
The OTP program on the phone, is guarded by a PIN. Choose a PIN that you can easily remember, but cannot be guessed by those who know you. Do not write it down.
OTP is used instead of passwords over non-secure communication lines or for exercise submissions. Since only the owner of the OTP generator can use it to generate the right string each time, OTPs are generally considered to be better authentication system than standard passwords (which can be easily reproduced, if stolen/overheard or obtained otherwise). They are something *you own* as opposed to something *you, and possibly someone else, know.*
Usually you'll need the OTP when submitting exercises or when connecting from outside.

1.2.2 Where and How to Logon?

You can log on into the student PC stations in Levy or Rothberg open space. Simply enter your login and password, and wait for a while.

Do not turn off the computers/screens. If you encounter problems, inform us, and we'll advice you on how to proceed.

[4]Students without a phone or other mobile device capable of running the OTP program, can receive OTP passwords via different methods. Please look at the password section in http://wiki.cs.huji.ac.il.

1.2.3 Disk Space

Each B.Sc. student receives 100 MB of storage on our filer + 100 MB of secondary storage (also known as the ".+" directory). This amount should be enough for your exercises as long as you don't fill it up with movies, mp3, games, etc. and as long as you clean up after you (i.e. remove temporary files). Later we'll discuss what to do when you disk gets full.

Unless you are a member of one of our labs, and need large disk space for your project, you probably won't get extra quota beyond these 200 MB. So try not to fill it up.

1.2.4 Backups

All the files under your home directory are backed up periodically several times a day. There are hourly, nightly and weekly backups which you can access directly. So if you accidentally delete something, you can recover it from an hour ago, 2 hours ago, etc.

The difference between the the primary and secondary storage space is the backup frequency. So try to use your primary storage as the working environment, where the secondary storage should be used for files that changes less frequently.

1.2.5 Printing

There are two types of printers for CSE students. One type are the *Mafil* printers, which are paid printers and are the printers in the students labs. The other type are the printers in Ross building which use printing quota and are not opened for B.Sc students.

1.2.6 web site

With your login, you receive web page in our servers which can be accessed from anywhere in the world from

```
http://www.cs.huji.ac.il/~<login>
```

The pages you put there are under your responsibility, and are part of your disk quota. Our server supports php, but only in secure mode.

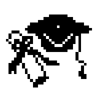
1.2.7 Mail account

Each student in the Hebrew University receives an email **account** under the domain: **mail.huji.ac.il** which is managed by google. This is the mail address where the university sends all its official mail.

Each student in the CSE school, receives another mail **address**: **<login>@cs.huji.ac.il**. This is only a mail address, and by default, every mail that arrives to this address will be forwarded to your **mail.huji.ac.il** account. It's possible to remove the forwarding and to use the cs.huji.ac.il mail as an independent mail account, please refer to the mail section in our wiki for instructions on how to achieve this.

1.2.8 Which Computers Can You Use?

Any computer in **Levi** and in **Rothberg** open-space. To use any other equipment you should be explicitly requested to do so by your TAs (Teacher Assistants).



From a Linux (PC) station, you will be able to work "remotely" on many other computers . The computer (server) intended for your use are called **river**. Use additional remote servers only if asked by your TAs.

1.2.9 Messaging: Web, Mail

Recommended Applications:

- for the Web: **iceweasel** (AKA **firefox**) to access the Web.
- for e-mail: As huji mail is a webmail, **iceweasel** should be used.

1.3 Exercises

1. Open an account as CS.
2. Register your OTP card. (hint: login as "register")
3. Log in to your account.
4. Send a mail to *penguin7-echo@cs.huji.ac.il*. In a couple of seconds, you should get an automatic answer from penguin7. Note: send it to *penguin7-echo*@cs, not to *penguin7*@cs.

1.3.1 Answers

1. Make sure you have your ID number (Teudat Zehut), also, choose a good login name and password.
2. Make sure you have your ID number (Teudat Zehut) available, and your cellphone powered and available.
3. Hope it works.
4. You can use huji.mail to send the mail (iceweasel).

Chapter 2

The Shell Textual User Interface

Textual Interface uses text (letters) to represent and manipulate files and programs instead of the more intuitive but limited Graphical Metaphors (icons, ...) used in GUIs.

Textual Interfaces are less intuitive and more difficult to learn. On the other hand they can be *ultimately more expressive:*

Lets imagine an Israeli tourist in India walking into a local restaurant. Not knowing the language, our friend the tourist must look around, point at someone's dish, grunt and point to his mouth and hope for the best. This is similar to using a mouse and icons. You cannot choose something that was not previously thought of - the tourist will probably end up eating something he didn't even want...

A more sophisticated Israeli might enter the restaurant, look at the menu in English, point at one of the dish entries, and hope for the best. This is similar to a interface of menus. This is exactly the same problem as before - the tourist has no way of telling the locals he don't not want too much hot pepper and curry in his lunch!

An intelligent tourist would bother to learn the language, explain to the Indian chef how to make Falafel, and make sure he doesn't put any curry and hot peppers in it! This is similar to a textual interface, you are actually learning a new language to speak to your computer!

Even though during your first year you may be able to get by only using grunts and pointing with your computer, it is certain that by the second year you will have to learn how to talk! We recommend you take time and learn the language now - you will be able to work faster and more efficiently even in your first year.

Especially useful are the **shell** programs. These programs let you type commands (programs) and by doing so, manipulate files, programs, networks, graphics, sound, etc.

UNIX Shells developed since its very first days - 30 years ago. They are much more powerful and expressive than any graphical user interface in existence. Their syntax allows us to **build complex commands out of simple ones** (like Lego blocks).

Shell is a **command line interpreter** (**CLI**). You type a command in a line, the shell interprets (understands) and executes it.

To work in a shell or any other program with textual input and output, you need a **textual terminal**.

2.1 Files, Directories

2.1.1 The File System Hierarchy

The file system is organized in a hierarchical structure of a tree. Where directories are it's branches, and files are it's leaves.

In reality each directory's contents may reside in a different location, a network drive, a remote computer, a local disk, or even RAM. This is transparent to us, the users.

Everything begins with the **root directory** '/'. The root directory is the only one without a parent.

If you want to specify a file, it can be done in a number of ways in the location field. For example, all of the followings are the same (for the file hello.c which is under penguin7's home directory, which is under stud, which is under cs, which is under the root directory):

when at penguin7's home directory:

- `/cs/stud/penguin7/hello.c`
- `./hello.c`
- `hello.c`
- `../penguin7/hello.c`
- `/cs/../cs/../cs/../stud/../../cs/stud/penguin7/././hello.c`

2.1.2 The Home Directory

Home Directory is the main place where user keeps her/his files.

In addition, all the applications keep their "user configuration" files there, e.g. if we configure the iceweasel to use a larger font, it will keep that information till the next time we use it in a file in your home directory.

In such way we completely **separate the dependency between users and specific computers**. Each user can work on any computer since all her/his work and configuration files are kept in the home directory. They won't interfere with someone else's configurations on the same computer.

Except for your home directory, you have another place where you can store (temporarily) files: `/tmp`. This directory cannot be used to permanently store files since it disappears each time a computer is rebooted.

Finally, you have more space, as large as your home directory, in which to store things, this is the directory ~/.+ which is located in your home directory /**cs**/**stud**/***login***/.+/ This is the perfect place for files that don't need hourly backups, such as objects that are generated from source code, large files that you don't have enough room for in your home directory, or old files that doen't change much. This area is backed up less frequently than the home directory, however, it is permanent storage and is not erased on a regular basis like /**tmp**.

2.1.3 Ownership, Permissions

If you have many users using a computer system at the same time, it is important to ensure that they cannot read each other's files, interfere with each other's programs, or interfere with the underlying operating system. This is referred to as system security.

This is very important to insure other students do not copy your exercises, modify them, or even erase them!

Ownership Each file/directory belongs to a certain **user** and **group.**

Permissions

- Each **file** can be accessible for:
  - **reading,**
  - **writing** and
  - **execution** [1]
- Each **directory** can be accessible for**:**
  - **reading**[2] (see the list of files in it),
  - **writing** (add a new file to it),
  - **changing into** (cannot be viewed or entered in any way).

Each one of these attributes can be relevant on three levels:

- USER - the owner of the file,
- GROUP - the group of users that own the file,
- OTHER - all the people that are not members of the group and have access to the system.

Later, we will discuss changing those permissions.

IT IS YOUR RESPONSIBILITY TO KEEP SENSITIVE FILES OUT OF OTHER PEOPLE'S REACH!!!

Directories in which you keep your exercise files are considered sensitive (e.g. `intro2cs`).

2.1.4 Groups

Basically, you *cannot* work on, or in some cases even look at someone else's files! In case you do want to cooperate with a group of one or more users and let them look at your files, you have to **create a group** to which they belongs using the 'group' command.

For example, to create a new group named 'mygroup' and add 'penguin7' to it:

```
execute 'group'
group> create mygroup
group> add mygroup penguin7
group> info mygroup
group> exit
```

You can then create a directory owned by that group, under which all files will be visible and writable to all the members (it sometimes take a couple of minutes for the group to get updated/created).

---

[1] each file can have a permission of an executable, but only files that contain programs/scripts can be run

[2] It just means that we cannot see the list of files in a directory, but if we know there is a file named X there we can access it, by explicitly stating X.

2.2 Processes

Process is a running instance of a program. There can be many instances of the same program running at the same time.

You can use the **top** utility to see the running processes. You can view, either your own processes, or all the processes in the system. They can be sorted (ordered) by different information (CPU utilization, memory utilization, time they are running, name, ...).

Note that each process has a **unique id**, called **process id** (**PID**)

If a process belongs to you, you can send it a signal. The most useful signals at this stage are **SIGTERM** and **SIGKILL:**

SIGTERM - ask the process to exit nicely (politely).

SIGKILL - murder the application without any prior notice.

You should always try SIGTERM before you murder it with SIGKILL, since it may want to save important data before exiting.

2.3 Basic Work in a Shell (tcsh[3])

Some of the commands are internal to shell (we order the shell to do something, e.g. change its state), and some are just programs with textual input and output, that are executed by the shell.

2.3.1 The Prompt

Prompt is a leftmost part of the line in which the shell informs you of certain parameters and notifies you that you are expected to enter a command.

Default for you contains (from left to right):

- command number
- exit status (how did the last command end: 0 is GOOD, $\neq$0 is BAD)
- your login
- the machine name
- the current working directory

When we finish typing a command we press **[ENTER]**

echo prints out whatever is given to it. *try*:

`echo hello world i am here` **[ENTER]**

2.3.2 Basic File/Directory Manipulation

`pwd` prints working directory (where am i?)

`pwd` **[ENTER]**

`cd` changes (current working) directory. *try*:

[3] There are several types of shells: sh, bash, zsh, csh, tcsh and more. The default here at cs is tcsh, but you can use whatever shell you like.

```
cd intro2cs
pwd
cd /cs/stud/penguin7/
pwd
```

- **paths** - chains of directories separated by /
  - relative paths (relative to the current working directory): `dir1`, `dir1/subdir`
  - absolute paths ("relative" to the root directory '/' - start at root!): `/cs/stud/penguin7/dir1`
- **special directories:**
  - **.** - the current directory
  - **..** - the parent directory. *try*:

    ```
    cd ..
    cd ../stud
    pwd
    cd .
    pwd
    cd ../stud/./././././././.
    ```

    ```
    cd
    ```
- **useful shortcuts**
  - ~ - your home directory. *try*:

    ```
    echo ~
    ```
  - ~**login** - login's home directory. *try*:

    ```
    echo ~penguin7
    cd ~penguin7
    pwd
    ```

    ```
    cd
    ```

`tree` displays file hierarchy. *try*:

```
tree intro2cs
cd intro2cs
tree
```

`ls` lists files. *try*:

```
ls
cd ..
ls
```

Arguments (**separated by space from the command**, usually starting with −):

```
ls -l     #(l)ong
```

- **lists files in the directory with relevant information for each file:**

```
ls -a     #(a)ll
```

➥ **lists all files, including hidden ones. (files starting with “.” like “.cshrc”; usually configuration files).**

```
ls -al    #(a)ll+(l)ong
```

➥ **lists all files, including hidden ones, with the relevant information.**

ls can get file-names and directories as arguments. *try*:

```
ls intro2cs Mail
```

`mkdir`, `rmdir` - creation (make directory), deletion (remove [an empty] directory). *try*:

```
mkdir hi
ls
rmdir hi
ls
```

`cp`, `mv`, `rm` - basic file manipulation (copy, move, remove). *try*:

```
mkdir practice
cp ~penguin7/hello hi
ls
cp ~penguin7/hello .
ls                          #notice what happened!
cp hi practice/hello
cd practice
ls
mkdir dir1
cd
```

In general form:

```
cp <name-of-file[s]-to-be-copied> <destination>
mv <name-of-file[s]-to-be-moved> <destination>
```

➥ Note: if there is more than one file to copy/move, the last parameter has to be a directory into which the files will be copied to: *try*:

```
cp hi ~penguin7/hello practice/dir1
cd practice/dir1
ls
cd                      # my head spins, jump back home
tree practice
mkdir greetings
mv hello practice/dir1 greetings # moving directories!
tree greetings
ls greetings                     # see the difference!
cd greetings
tree
ls
cd ..
rm greetings                     # error!
rm -r greetings
```

```
cp practice practice.backup     # error!
cp -r practice practice.backup
ls
tree practice.backup
```

2.3.3 Commands Dealing With File Contents

create your own file:

use 'echo' to create a file. *try*:

```
echo hello, world! This is my file > my-file
ls
```

`cat` concatenates files or standard input to standard output. *try*:

```
cat hi
cat practice/hello
cat hi practice/hello
cat                          # write your message and finish with ^C
```

`less` shows file contents (screen after screen). *try*:

```
less my-file
less ~penguin7/hello         # quit with 'q'
```

- ➥ you can **search for strings** in the file by using /<word-you-are-trying-to-find> **[ENTER]**
- ➥ you can repeat the previous search by pressing '**n**'
- ➥ you can search backwards by pressing '**N**' (that's **shift** + **n**)

`grep` - search for string in files and return the entire line. *try*:

```
grep everyone ~penguin7/hello
```

`diff` - find differences between files. *try*:

```
diff hi ~penguin7/hello
diff my-file hi
```

`touch` - "touch" a file - set the file modification time to the current time, without really changing the file. *try*:

```
ls -l hi
touch hi
ls -l hi
```

- ➥ if file doesn't exist, create it with the current time. *try*:

```
touch A B C
ls -l A B C
```

`wc` - word count. *try*:

```
wc hi
       #shows: number of lines, of words, of letters
wc -w hi
       #shows: number of words only.
```

2.3.4 Changing Permissions

`chmod` changes permission mode.

try:

```
ls -l hi
chmod a-w hi        # removes write permissions from all
ls -l hi
chmod go-r hi       # removes group's, other's read permissions
chmod g+r  hi       # adds read permissions to group
chmod ug+w hi       # adds write permissions to user and group
chmod u+x  hi       # adds execution permission to user.
chmod g-w           # don't leave group/other writable files/directories
```

if you want to create a directory under which all files are accessible for both reading and writing to your group members type:

```
chmod g+rwxs <file-name>
```

Important: Do '`chmod go-rwx <your-course-exercise-directory>`'

2.3.5 Basic Process Control

Every time you run a program it is given a unique[4] number. We call this number a process ID. It is used to identify your program in an event you would like to control it - for example stop it from running. Some programs start multiple processes, for example iceweasel.

- by default the shell **waits** for a program to end, and does not let you enter more commands till it is done:

```
xeyes            # close window manually to release the shell
xeyes                # close with '^c'  = [CTRL-C]
```

- to continue working, shell **does not wait** for a program to end, use:

```
xeyes &              (background mode)
```

- **suspending** processes, sending to background, foreground. *try*:

```
xeyes                # suspend with '^z' [CTRL-Z], xeyes sleeps
fg               # foreground, suspend again '^z'
bg               # background
```

- The shell assigns a job ID for each program run from it. The job ID is unique **per shell** only.

```
xdaliclock &
xeyes &
jobs             # list of jobs that we started
kill %2          # kill job number 2
fg %3            # job number 3 to foreground
```

[4]This number is unique at a certain time on a certain computer. For example your program may have the same process number as your friends if you are working on different computers (and are very lucky).

- processes in the system:

```
top            # interactive list of process. q to quit. k to kill a process.
ps             # list of processes
```

PID System Wide **P**rocess **ID**

TIME time since the process was started

CMD Command Name

try:

```
ps x
ps ux
ps uxf
ps uxfa
kill <PID>   # kill a process by a global PID, not a job number
```

- **Note:** Kill means send a signal

```
kill -KILL <PID>        # murder the process
kill -TERM <PID>        # ask it politely to die
```

```
xeyes &
xeyes &
xeyes &
xeyes &
killall xeyes             #a shortcut
xeyes &
xkill      # now click on the xeyes window
```

- You may use the command top in order to list the various processes control and kill them.

2.3.6 Printing Commands:

`a2ps` prints files: `a2ps <file-names>`
`a2ps -P<printer name> <files...>`

lpr also prints files (not so nicely)

`lpq` shows the jobs in the printing queue.

`lprm` removes a job from the queue: `lprm <job-number>`

In the students computer lab all printings are directed to the *Mafil* printers (public servers default to *Mafil* printers).

To print on other printers (for those who have access), most printing commands receive the "-P<printer>" parameter and they'll send the printing to the given printer. Another option is to set the environment variable PRINTER (see 5.2 on page 43) before executing the command that's about to print. For example:

```
env PRINTER=xl1 ooffice
```

Will make the next file you'll print from open office, to come out in xl1.

It is best to print using a2ps, or mpage, as they can easily print 2 or more pages on one paper. a2ps also has some nice features for printing all kind of documents.

Sometimes, you'll need to print from a graphical program (iceweasel, openoffice, gv, etc.), and if you want to print with special parameters (e.g. mpage -2, which prints 2 pages per sheet, or a different printer) you'll have to enter the command on the "print command" prompt in the print dialog. If not possible (like in open office), either change the environment variable PRINTER as stated above, or simply "print to file" and than print the ps file any way you like.

Sometimes, a print job get stuck and/or refuses to print. If that happens, don't leave your print job there, because it'll make the printer unavailable for others. Instead, check if it's in the printing queue with **lpq**, and if so use **lprm** to remove it.

2.4 Efficient Work in Shell

2.4.1 Expansion & Globbing: ~ * ? [] {} ‘‘

~ expands to a home-directory path

***** expands to any string within a file name

? expands to any character within a file name

{abc,def,qwert,...} expands to either one of the strings: abc, def, qwert, ...

[abc] character class - expands to either one of the letters in []. Characters can be given in ranges [a-zB-K].

‘command‘ expands to the output of the command.

\\~,*,\\? expands to ~, *, ? accordingly

***try*:**

```
echo hello
echo ~
echo *
echo \~
echo \*
touch aua aba aca bbb aqua aquarel
ls
echo a*a
echo a?a
echo *a
echo *ua
echo a*
ls -l a*
echo aq*a
echo aq*a*l
echo /cs/stud/i*
echo /cs/stud/i*/a*
echo ?b?
echo ?b*
echo a[cb]a
echo a{c,bracadabr}a
echo a{caca,braca/da/br}a
mkdir all_the_As
mv a* all_the_As
```

‘<command>‘ executes the command and expands to its output.

***try*:**

```
pwd
echo current directory is `pwd`
```

2.4.2 History

```
history        # prints the command history.
!<comm-num>    # executes the command by its number from history.
!!             # executes the last command.
[↓] [↑]        # browse through history.
ls [ALT-P]       # completion from history, last command begin-
ning with 'ls '
!$             # expands to the last argument of the last command
```

***try*:**

```
ls
history
!!
!1
!-2
touch hi
ls -l !$
cat !$
```

2.4.3 Command Name, File Name Completion

[TAB] completes a command, file name, variables. We have to type a letter to resolve the ambiguity:

- **File Name Completion (for command arguments - not the first word in the line).** *try*:

  ```
  cat ~penguin7/s[TAB]              # gets the list of files start-
  ing with 's'
  cat ~penguin7/sup[TAB]          # gets a nar-
  rower list, of files starting with 'sup'
  ```

- **Command Completion (if tab is used at the beginning of the line - command).**

try:

```
m[TAB]
mo[TAB]
moz[TAB]
mozill[TAB]
```

2.5 Using external storage media

On computers with usb connection, you can attach a usb storage device and it will automatically appear in a window. You can access it in the shell via /media directory.

When finished working with the disk, you should unmount the disk before removing it, otherwise your data might get corrupted. To do so either use right-click the disk icon and select unmount, or use the udisks command (see our wiki for further help on this issue).

2.6 Miscellaneous Important Commands

2.6.1 Disk, Printer Quotas:

- on disk space:

  `nquota`, (See 9.3)

- on printers (for Ross printers, for people who have quota)

  `stquota`

2.6.2 On-Line Help, Documentation

Command "Self-Documentation", a short description of options and synopsis

```
<command> --help
<command> -h
```
try: `ls --help`

Manual Pages:

The UNIX Manual pages (or "man pages" for short), are difficult to understand in the beginning. Don't get discouraged, once you understand **how** to read them, you will find them more helpful and comprehensive then other help systems. In order to get used to them, take a look at a man page of a command you know well, for example 'man cp'.

- To get the man page of a command or function:

  `man <command/function/...>`

- To search for man pages related to a keyword:

  `man -k <keyword>`

- Each manual page consist of sections like: NAME, SYNOPSIS, DESCRIPTION, ..., SEE ALSO - sometimes you only need to read a single section to find the information you need.
- There is usually no need to read the entire man page in order to find a single switch in a command. Use the **search** command in your pager (/) in order to find the switch you need.
- They are divided into sections: 1-9; The important ones are: 1 for commands, 3 for C library functions and 2 for system calls
  - "`man 1 printf`" vs. "`man 3 printf`"
- You can also use **emacs** to browse man pages, this may be an easier way to navigate.

Info Pages

Are like online books, e.g. description of Makefiles

- info <command/subject>

  or

- **CTRL-h-i** in Emacs.

HUJI-CSE On-Line Documentation

```
http://wiki.cs.huji.ac.il
http://www.cs.huji.ac.il/ --> click on online support
```

2.6.3 English Dictionary and Spallllllung (Spelling)

- Under emacs you can spell-check and complete words from English dictionary [ALT-$] and [ALT-?]
- To print English words starting with <word>:

```
look <word>
```

- To look up a a word in the dictionary:

```
dict <word>
```

try:

```
look antidis
dict abominable
dict `look abomin`
```

2.6.4 Other Useful Commands

```
locate <string>      # fast find files
cal                  # a calendar
bc                   # a calculator
tail <file>          # print the "tail" of a <file>
head <file>          # print the "head" of a <file>
repeat <n> <cmd>     # repeat <cmd> <n> times
```

```
    repeat 10 echo ``happy, happy, joy, joy''
```

```
sort [<file1> ...]   # sorts stuff
```

```
    sort ~penguin7/intro2cs-students
```

- # **sorts file in lexicographic order**

```
    ls -l | cut -d' ' -f5- | sort -n
```

- # **sorts files by size ('-n' numerical order, not lexicographic)**

```
passwd          # change your password
date            # print the current date, time
who, w          # who's there on the computer
ln -s           # create a "symbolic" link - a shortcut
```

try:

```
touch hello
ln -s hello hi
ls -l hello hi
```

- # **creates a symbolic link ("shortcut")** `hi` $\rightarrow$ `hello`

```
find     # can do anything on files, read the man page
```

```
    find . -name '*.c'
```

- # **finds files ending with .c in all sub directories of current directory**

```
find ~penguin7 -size +100
```
➥ # **finds file larger that 100KB in penguin7's home directory**

```
find . -ctime -10
```
➥ # **finds files created in the last 10 days**

```
find . -ctime -10 -name '*.c'
```
➥ # **finds C source files created in last 10 days**

```
...
```

`alias` # **setup aliases for commands:**

```
alias  mydir 'ls -l'
```
➥ # `mydir` **command is now** '`ls` **-l' in disguise**

```
mydir
```

2.6.5 Other Less Useful Commands

Some of these commands are under /usr/games. For those commands to work, the PATH environment variable (see section 5.2 on page 43) will have to be set to include '/usr/games' or the full path of the command will have to be specified.

```
tac          # like cat only reverse
cowsay       # cow says interesting things
number       # prints the numbers
pig          # Pig Latin
fortune      # fortune cookie
quiz         # quiz
sl           # learn not to make mistakes while typing 'ls'
figlet
banner
hello
vi
```

2.7 Exercises

2.7.1 basic shell

1. How many files are under ~intro2cs home directory?
2. How many files and how many directories are in your home directory? How many hidden files and directories are there?
3. What is the biggest file (including hidden ones) in your home directory? How big is it?
4. What is the full path of your home directory?
5. How large is your home directory? hint: du
6. Create a directory ~/i2u/ex1
7. Create the following directories under ~/i2u/ex1. Using the '-p' option (see the man page) you can do it in four commands only (and even a single command), assuming you are in the directory ~/i2u/ex1.

```
1
|-- 2
|    |-- 3
|    `-- 4
|
`-- 5
     |-- 6
     `-- 7
```

8. How can you get to penguin7's home directory, no matter what your current dir is? Explain two ways.

9. Copy the file ~penguin7/hello to ~/i2u/ex1/1/2/

10. Copy the same file, ~penguin7/hello, from the previous question, but this time, copy it to a file called hello2 in the same location of ~/i2u/ex1/1/2/

11. Change directory to: ~/i2u/ex1/1/2/ and copy the hello file in there to the parent directory (~/i2u/ex1/1/). What is the shortest command?

12. Change directory to: ~/i2u/ex1/1/2/ and copy the hello file in the *parent* directory (~/i2u/ex1/1/) to your home directory - but change it's name to hello3.

13. Rename ~/i2u/ex1/1/2/hello to ~/i2u/ex1/1/2/hi

14. Copy all the files under ~penguin7 to ~/i2u/ex1/

15. Move the files *hello.c* and *hello* from ~/i2u/ex1/ to ~/i2u

16. Copy the entire directory structure under ~/i2u to ~/i4u

17. Now delete all the files/directories created inside ~/i2u/ex1

2.7.2 Permissions

1. Given:
   —-rwx-wx 1 penguin7 stud 20 Feb 23 19:34 /tmp/file1
   what actions can penguin7 do? What can another student do? What can someone else do?

2. How can penguin7 change it to:
   -rwx-w-r-x 1 user1 group2 20 Feb 23 19:34 /tmp/file1

3. Change the permissions of ~/i2u/ex1 directory so no one can read or write to it apart from you.

2.7.3 Man and Help

1. How many days are in September in the year 1752? How many days are in September 2005?

2. How many sections are in 'man'? What section/s is/are related to 'C' functions.

3. Both section 1 and 3 describe different commands called 'printf': What section is shown when you 'man printf'? How can you get the other? How can you know to what section belongs a manage you are looking at?

2.8 Answers

Note that in general, there is more than one possible answer. Here we show the shortest/advanced one. The '#' sign, means the beginning of comments, the rest of the line isn't a part of the command.

2.8.1 basic shell

1. ls ~intro2cs # or cd ~intro2cs ; ls
2. Move to your home directory using *cd*, then try
   (a) ls -l # non-hidden files and directories
   (b) ls -a -l # **a**ll files and directories, hidden and non-hidden.
3. Move to your home directory using *cd*, you can use the *-S* option in *ls*, to get it **s**orted.
   (a) ls -la -S # or ls -Sla or ls -aSl or ls -l -S -a or .....
4. cd
   pwd
5. For a **s**ummary size of files under the directory: du -s ~/ or for a more **h**uman readable output du -sh ~/
6. Two ways:
   (a) mkdir ~/i2u; mkdir ~/i2u/ex1
   (b) mkdir -p ~/i2u/ex1
7. There are many solutions. This solution is the faster and uses *mkdir* only.
   (a) cd ~/i2u/ex1/
   (b) mkdir -p 1/2/3
   (c) mkdir 1/2/4
   (d) mkdir -p 1/5/6
   (e) mkdir 1/5/7
   (f) You can use *tree ~/i2u/1* to see how it looks.
8. Both this commands will change your directory to penguin7's home dir.
   (a) cd ~penguin7
   (b) cd /cs/stud/penguin7
9. cp ~penguin7/hello ~/i2u/ex1/1/2/
10. cp ~penguin7/hello ~/i2u/ex1/1/2/hello2
11. cd ~/i2u/ex1/1/2/
    cp hello ..
12. cd ~/i2u/ex1/1/2/
    cp ../hello ~/hello3

13. mv ~/i2u/ex1/1/2/hello ~/i2u/ex1/1/2/hi
14. cp ~penguin7/* ~/i2u/ex1/
15. mv ~/i2u/ex1/*hello** ~/i2u/
16. cp -a ~/i2u ~/i4u
17. rm -rf ~/i2u/ex1 # beware rm -rf is a dangerous command

2.8.2 Permissions

1. Penguin can neither read, write, nor execute file1
   Other students may read write and execute file1
   Everyone else may write and execute file1, but not read it.
2. Two ways:
   chmod 725 /tmp/file1
   chmod u+rwx /tmp/file1; chmod g-rx /tmp/file1 ; chmod o+r /tmp/file1; chmod o-w /tmp/file1
3. The long way:
   (a) chmod u+wr ~/i2u/ex1; chmod g-wr ~/i2u/ex1; chmod o-wr ~/i2u/ex1

2.8.3 Man and Help

1. Try[5]
   (a) cal 9 1752
   (b) cal 9 2005
2. There are at least 8 sections. You can use 'man man' to find out about man including each section, as well as reading, for example, 'man 2 intro' to get an introduction on section 2. The upper-left corner of the manpage, shows the manpage name and section number. For example: CAL(1)
3. The first section having the command is shown. You can choose section 3 (or any other if exist) by *man 3 printf*. The section number appears in the upper-left corner.

[5] The Gregorian Reformation is assumed to have occurred in 1752 on the 3rd of September. By this time, most countries had recognized the reformation (although a few did not recognize it until the early 1900's.) Ten days following that date were eliminated by the reformation, so the calendar for that month is a bit unusual.

Chapter 3

Emacs

Emacs is one of the most used editors in UNIX environment. It will probably be your closest friend for the next few year, so you'd better learn how to use it and benefit from it. It is very powerful and complex tool and you will never know all it can do. BUT:

You can easily learn how to edit your files using the different menus of emacs.

There are also many short-cuts to help you save time and make work easier for you.

- To enter Emacs type: `cemacs <file-names...>` (this will open the file in an already opened Emacs if available, to open a new Emacs - not recommended - execute `emacs`)
- To open a file: **CTRL-x CTRL-f**.
- To incrementally search for a string in the file: **CTRL-s**.
- To save a file: **CTRL-x CTRL- s**.
- To compile your program press **F9** (make or javac *.java)
- Next compilation error: **F10**.
- Speedbar: **F12**
- Spell-check word under cursor: **[ALT-$]**
- To complete from the dictionary: **[ALT-?]**
- To complete from previously typed words: **[ALT-/]**
- **[TAB]** - for perfect indentation (it is strict)
- **[SHIFT-F1]**: for man page of the word under the cursor.
- **CTRL-k** cut from the cursor to end of line. Copy it to the ring/buffer.
- **CTRL-Space** mark the start of a region. Then use the arrows to move.
- **ALT-w** copy the region to the ring/buffer.
- **CTRL-w** cut the region to the ring/buffer.
- **CTRL-y** paste the contents of the ring/buffer.
- You can reach many other options by **ALT-x** and writing their names (for playing TETRIS: **ALT-x** tetris...)

- ➩ **[ALT-x]**flyspell-mode, in which case any misspelled word will be underlined

- ➥ To quit Emacs: **CTRL-x CTRL-c**

You can do a lot of neat stuff with emacs. You will have to play a little bit to find out...
If you want to learn more about Emacs you can refer to the tutorial at HELP of Emacs (**CTRL-h t**).
Also, it's a good idea to print the emacs reference card located at `‘‘/usr/share/emacs/22.3/etc/refcard.ps` or in our wiki, and keep it handy for your first couple of weeks.

3.1 Exercises

1. copy three fortune-cookies to a file called bestFortunes (you can use the mouse for copy/paste)
2. Using 'emacs' (use the copy and paste shortcuts, not the mouse), create a file "tigres.txt " containing 4 times the following two lines:

   Tres tristes tigres comen trigo en un trigal.

   Que rapido ruedan las ruedas del ferrocarril.
3. Use the diff command to check it is the same as ~penguin7/tigres.txt. Hint: use *diff -b*.
4. Change one letter in ~/tigres.txt and use the diff command again.

3.2 Answers

1. /usr/games/fortune # Three times, to output three fortune cookies. We hope you used the mouse...
2. Using the proper cut and paste shortcuts.
3. diff -b ~penguin7/tigres.txt ~/tigres.txt # there should be no output
4. diff -b ~penguin7/tigres.txt ~/tigres.txt # now diff will give output

Chapter 4

Work “On Remote”

Network: In order to better utilize dispersed computer resources and to enhance human communication, we connect computers into ***computer networks***. Once we have a ***physical connection***, computers can talk to each other by using different “*protocols*” - languages for computer/application inter-communications. Here at CSE all of our machines are physically interconnected into a “*local area network* (**LAN**)”, which enables them to talk among themselves. This is great, since then we can do all kinds of neat tricks, like mapping `/cs/stud` from all our computers to the same place on the network.

Internet The ***Internet*** is a global network consisting of many Local Area Networks. All the computers of the Internet use the same protocol, called the ***Internet Protocol*** (`IP`). This enables them to talk to each other.
Our (CSE) Local Network is an integral and permanent part of the Internet.

Work “On Remote” is the *ability to run applications on another (REMOTE) computer*. To do that, your workstation (computer) has to be connected to that computer through a network. In theory you can work remotely on any computer on the Internet, if it supports remote access (e.g. UNIX systems). In practice, not many computers will allow you to do that (you don’t have an “account” on them).
You can work remotely on many of our (CSE) computers: river, inferno ...

Why Work On-Remote? Different computers have different resources. There are super-computers that can do very fast computations. There are computers that have very large and fast disks, which makes them excellent as data storage devices. Others have very good networking capabilities... In order to best utilize all these resources we have to allow many people/computers to work on them at the same time. Otherwise, most of the time they will be idle and the expensive resources would be wasted.
During your studies you will use a lot of different hardware (PCs, SUN Sparcs, ...), which run a lot of different Operating Systems: Solaris, Linux, FreeBSD, ... Each one of them has certain advantages over the others. You will be able to fully explore the advantages by working on them remotely.

4.1 Work On-Remote Using Textual Interface

To connect to a different computer and work on it using a textual interface (e.g. shell), you can use the following command:

```
rlogin <computer’s-internet-name>
```

- This command (remote login) will make a connection to a remote computer and redirect all the input/output between the textual terminal and the remote computer.
- On the "server" side a shell program is started. since all of our input/output is redirected to there, we are in fact working on a shell that is running on the remote computer

To execute a remote command

```
rsh <computer> <command>
```

- this will execute <command> on <computer>. *try:*

```
hostname
rsh river hostname
```

- by default if no `<command>` is given, rlogin is executed.

```
rsh river
```

- for a secure shell you can use ssh instead of rsh (in some cases you won't be able to use rsh so ssh will be your only option), however rsh is preferred inside the CS network.

4.2 Work On-Remote Using the X Windows System Interface

The X Windows System enables you to run any Graphical application on remote.

The "X Windows Display Server" is a process that runs on your machine.

It takes the control of your:

- Input: Keyboard, Mouse,
- Output: Screen.

Any application that wants to draw something on your screen or get the input from your keyboard, has to ask the "X-Display Server" to do it. The application cannot do it directly!

The actual input (keyboard, mouse) and output (drawing on screen) is done by the X-Display Server on behalf of the application.

This approach makes it irrelevant whether the application itself is running on the same computer as the X-Display Server or a different one. A remote application can connect to your Display through the network, and you will not know the difference.

```
iceweasel&           # run a firefox on the local machine
rsh river iceweasel --display os-61:0
        # run mozilla-firefox on river with output on os-61
             #  (we assume you are working on os-61)
```

- a shortcut for this is:

```
xon <computer> <graphical-command>
```

try:

```
xosview&               # resource usage statistics
xon river xosview
xon inferno xosview
```

➥ by default, if no `<graphical-command>` is given, '`xterm`' is run.

try:

```
xon river
```

Note: `.rhosts` file contains a list of machines (computers) *from which you can initiate* rsh and xon command to other hosts. See section 6.1 on page 49 for more information.

4.3 File Transfer

FTP File Transfer Protocol, can be used to transfer files from/to remote computers

```
ftp <machine-name>
  ➥ # connect to a remote machine for file transfer
      help [<command>]     # print a help on commands
      ls                   # list files in the remote directory
      cd  <directory>      # change to a remote directory
      lcd <directory>      # local change to directory
      get <file>           # transfers a remote <file> to local disk
      put <file>           # transfers a local <file> to remote disk
      mget <file-pattern>  # multiple get
      mput <file-pattern>  # multiple put
      pwd                  # print remote working directory
```

4.4 Working from outside the CSE network

Connection to the servers is also possible from outside the school's network, e.g. from a personal computer at home. However, the appropriate programs needs to be installed. All connection from outside must pass through a speical server named: **gw.cs.huji.ac.il** (gateway). A connection from outside requires an OTP password for the connection to gw, and the normal password for connecting to a proper server.

4.4.1 Textual interface

In order to connect through gw.cs.huji.ac.il, the actual server and the username must be specified. So the connection will be to: **<username>%<host>@gw.cs.huji.ac.il**. For example:

```
ssh penguin7%river@gw.cs.huji.ac.il
```

The proper program to display and connect needs to be installed. On Linux or any type of Unix or on Mac OS X, it's probably already installed. For Windows, an additional program is needed, e.g. *putty*.

4.4.2 Graphical interface

For forwarding the graphical display, the local machine needs an X window server to display the remote program's windows. For Linux and Unix, it's already installed. For Mac OS X, it's an optional installation (the *X11* program), and for Windows an additional program need to be downloaded and installed (e.g. *Xming*, or *Cygwin*). On Linux and Mac, assuming the X is running, the simplest way of connecting is with the -Y option of ssh. e.g.:

```
ssh -Y -C penguin7%river@gw.cs.huji.ac.il
```

Note that forwarding display through home internet connection can be sometimes very slow and somewhat annoying for long work.

On Windows, the X server needs to be active and the connection should be told to forward X11 data. This can be done by e.g. running *Xming*, and specifying in *putty* to forward X11. Complete instructions are in the wiki: http://wiki.cs.huji.ac.il/wiki/Connectin

4.4.3 File transfer

Any text based ftp client can be used to connect to **gw.cs.huji.ac.il**. After entering the username and the OTP, execute the **user** command with **<username>@<hostname>** parameter, in order to connect to an ftp server. e.g.

```
user penguin7@river
```

Graphical based ftp client might also work. But they need to either support textual connection method (to execute the **user** command), or a proxy connection to use gw.cs.huji.ac.il as a proxy. Several examples for graphical ftp clients' configurations can be found in the wiki: http://wiki.cs.huji.ac.il/wiki/FTP

4.5 Exercises

1. Do the man pages in river and bagel differ? Try *man cal* in both computers.

2. Does firefox differ on inferno and bagel?

3. Do you think there are other differences? What about libraries versions and paths?

4. Open xeyes in a friend's display. Does he need to authorize it? How?

5. Use FTP to get the file /pub/linux/Mdk/mdk91/README, from the server called ftp.cs.huji.ac.il. Using *lcd*, store the file under your ~/tmp/ directory (if doesn't exist yet, create it **before** running ftp). Note, you must login as *anonymous* and your password is a your email (e.g.: penguin7@cs.huji.ac.il).

6. Using the *df*[1] command, check where /usr/local/bin and your home directory came from.

7. Use *talk* to use the grandfather of the ICQ/Messenger, and communicate with a friend in another computer inside CS.

8. Remove your current machine from your .rhost, and try *rsh river*. What happens with *xon river*? Why?

[1] man df

4.5.1 Answers

1. Yes, they do. This is because the operating system running on bagel and river differ (although both are unix). Moreover, even running an OS from different distributions/flavors or versions make the machines differ.

2. Yes they do. As in question 1 answer, the firefox in bagel and river, are from different versions.

3. Different OS flavors/distribution/versions, put they files in different places. When you learn about makefiles, this will be an important issue. Generally, you can use *locate* to know where a file is (i.e.: locate stdio.h).

4. Your friend needs to run *xhost+* `<your machine>`. The display is like a file, you need other's authorization to write to it. Be carful with this command as it will grant anyone working on the other machines access to your display.

5. ftp ftp.cs.huji.ac.il # See Section 4.3 on page 37

6. *df /usr/local/bin; df ~/*
   In this example:
   df /usr/local/bin ~/
   Filesystem 1K-blocks Used Available Use% Mounted on
   nafs-02:/dist 57671680 45085856 12585824 79% # /usr/local/bin, came from a server called nafs-02
   /dist fs-*01*:/stud 36700160 28294656 8405504 78% /a/fs-01/stud # Your home dir came from a server called fs-01

7. Assuming your friend penguin7 is logged in river: *talk penguin7@river.*

8. rsh asks you for your password.
   xon doesn't work.
   Check the ~/.rhost file in section 6.1 for an explanation.

Chapter 5

Advanced Shell

5.1 Redirection & Piping

stdout Standard Output. Regular output of your programs, usually on the text console.

stderr Standard Error. Error (indicating) output of your programs, also to the text console.

stdin Standard Input. Textual input to your program, usually from the keyboard.

Each one of these three basic `I/O` (Input/Output) sources, can be *redirected* - passed to/from a file, or to another program.

5.1.1 Redirection in Shell: '<', '>'

To redirect `stdout` (and `stderr`) of command `<command>` to a file `<file-name>`:

```
<command> <arg1> <arg2> ... > <file-name>
```

➥ if the file exists, overwrite it using:

```
<command> <arg1> <arg2> ... >! <file-name>
```

➥ or append to an existing file using:

```
<command> <arg1> <arg2> ... >> <file-name>
```

➥ to catch both `stdout` and `stderr`:

```
<command> <arg1> <arg2> ... >& <file-name>
```

try:

```
echo hello > hello_file
ls
less hello_file
```

➥ **# instead of the console, the output of echo ("hello") is send to file hello_file**

```
cat
```

➥ **# cat without arguments, copies stdin to stdout (CTRL-D for End Of File)**

```
cat > file1
```

➥ # **stdin from cat to file1**

```
less file1
cat file1
```

➥ # **cat followed by file-names, prints their contents to stdout, one after another**

```
cat file1 > file2
ls
```

➥ # **instead of console, prints the contents of file1 to file2**

```
echo hello > file1
```

➥ # **ERROR: file1 already exists, cannot redirect to it**

```
echo hello >! file1
less file1
```

➥ # '>!' **forces overwrite (previous contents of file1 will be replaced by 'hello')**

```
echo hi >> file1
```

➥ # **'hi' will be appended to file1**

```
cat file1
```

To redirect input from a file to `stdin` of a command:

```
<command> <arg1> <arg2> ... < <file-name>
```

try:

```
cat < file1
```

➥ # **redirects the contents of file1 to stdin of command 'cat'**

```
wc < file1
wc file1
echo ``in file hello there are `wc -w < hello` words''
```

Example of combined redirection:

```
ls -l > list_of_files
grep 'rwx' < list_of_files > list_of_executables
cat list_of_executables
```

5.1.2 Piping in Shell: '|'

Piping is a way of passing the output (stdout) of one program directly to the input (stdin) of another:

```
ls -l | grep '-rwx'
```

try:

```
ls -l | wc -l
```

➥ # **how many files are there? (`wc -l`, counts the lines)**

```
echo 'Hi, penguin7' | mail penguin7@cs
```

➥ # **sends mail to penguin7 with the message 'Hi, penguin7'**

```
ls /cs/stud | sort -r
```

➥ # **gives student list in reverse alphabetical order**

```
pwd
pwd | cut -d / -f 3-4
```

➥ # **prints the 3rd and 4th component of your working directory path**

```
ps axu | grep penguin7 > penguin7s_processes
```

➥ # **all processes belonging to penguin7**

```
ls -l | grep '^-rwx'
```

➥ # **all executables**

Example - 'killall xeyes' simulation (for OS-s without killall command)

```
kill `ps ux | grep xeyes | grep -v grep | awk '{print $2}'`
```

Remember the '`alias`' command, it comes handy together with the above.

5.2 Shell Variables, Environment Variables

Shell Variables:

try:

```
set greeting = hi
echo ``$greeting from kreso''
set exdir=/cs/stud/penguin7/intro2cs
cd $exdir/ex1
```

Environment Variables are like shell variables, but are "***inherited***" when a new process is created - child process "knows" them.

try:

```
set name = penguin7
setenv a_better_name kreso
```

➥ # **this is how we define environment variable in tcsh shell**

```
xterm
```

➥ # **in the new xterm type:**

```
echo $name              # unknown variable
echo $a_better_name     # (you know which name is better)
```

```
printenv                              # prints all env. variables
```

5.2.1 Useful Environment Variables (IMPORTANT)

`DISPLAY` - contains a machine running X Display Server, to which graphical applications direct their input/output by default (usually the computer you are currently working on).

try:

```
telnet <remote-machine>
  ➥ # on the remote machine, type:
    xclock
          # ERROR: don't know which display to send graphics to
    xclock -display <machine-youre-sitting-on>:0
          # explicitly state display
  setenv DISPLAY <machine-youre-sitting-on>:0
    xclock
          # knows where to send graphics to, by looking at DISPLAY env. variable
    xeyes
```

`CLASSPATH` contains path list where JVM (The Java Virtual Machine) looks for classes. By default, this env. variable is defined in **.classpath** file.

`PATH` contains list of paths where the Operating System (UNIX) looks for executables:

```
hw
    # ERROR: command not found -
    trying to run a program (executable) by name 'hw'
echo ${PATH}
setenv PATH ${PATH}:/cs/stud/penguin7/bin
echo ${PATH}     # see the new contents of PATH
hw
    # now it works, since there is such program in /cs/stud/penguin7/bin
```

`PRINTER` default printer name (not relevant for *Mafil* printers)

```
echo hello | lpr -Pst # we have to state the printer explicitly
setenv PRINTER st      # we define st to be our default printer
echo hello | lpr       # will send hello to printer st
```

➥ # **this env. variable is predefined in .cshrc file (See 6.1).**

5.3 Scripts (Interpreted Textual Programs)

Scripts are programs in a very high language, e.g. tcsh (shell) or Perl.

The first line (she-bang `#!`) contains the name of the interpreter.
When Unix sees `#!` at the beginning of an executable text file[1], it runs the interpreter on the file itself.

Note that the file has to have executable permissions! This mechanism is general, and enables writing of scripts in any interpreted language.

The default shell here is tcsh, so if you want to use several commands on after the other many times (usually for checking your exercises), you can simply place them in a text file with the first line as: "`#!/bin/tcsh`". And then run it as many times as you like. you can use the special variable $1, $2, etc. to find out what the arguments of the script were.

Shell scripts can be far more complicated than executing simple commands one after the other, but this is beyond the scope of this introduction.

[1]The file has to have executable permissions (`r-x`, do a '`chmod +x <file-name>`' to set them)

5.4 Exercises

1. How can you know where a file called *fortune* is, without using *find*? Hint: see section 2.6.4.

2. What happens if you try to execute a file that exist somewhere in the system, but no file with that name is in your path? Why? How can you solve this? Try executing *fortune*.

3. How can you add a directory to your PATH? Add /usr/games/ to your path.

4. Copy 3 fortune-cookies to a file called bestFortunes. Use redirection (> and >>). If you get *fortune: Command not found* error, check the answers for the questions 1, 2 and 3.

5. Create a hard link to bestFortunes called hardL . Create a soft link to bestFortunes called softL. Hint: *man ln*

6. What happens if you delete bestFortunes: will *cat hardL* still work? What about *cat softL*?

7. What is the difference between hard-link to a file and copying it? What happens if we edit/modify a hard-linked file? What if we edit/modify a copied file? Does the hard-link occupy space? Try with the *du*[2] command.

8. Using pipe '|', *ls* and *wc*: write a command that show the number of files in the current directory.

9. Write a script that get 2 arguments, and prints them in different order.

10. Write a script called backing.csh that get one argument, and makes a backup of all the files meeting the pattern. For example if the files *a.c, b.c* and *c.c* exist, then *backing.csh '*.c'* will copy *a.c* to *a.c.back*, *b.c* to *b.c.bak*, etc. Note that a file a.cc nor any other file that doesn't meet the pattern must not be backup'ed. Why are the tag (') surrounding the pattern necessary when calling *backing.csh?*

11. Use *grep*, to list all the files in you home directory, that set/modify your CLASSPATH environment variable. What happens if you use *grep -i* ?

12. Write a script that finds all core files prints their full path, and asks if you want to erase them.

13. Read man grep, write a script that searches all *.h files under /usr/include for a certain word.

14. Make a shell script that does kill -9 to all firefox processes

15. Convert all shell scripts to aliases (if possible).

16. write an alias that erases all files *~, *.bak, and #*#

17. assuming pine uses ~/.addressbook for it's addresses write a shell script that prints a given user's address

18. use complete in various ways

19. read man page of gv, a2ps, mpage, lpr, print out homework nicely (stquota)

[2] man *du*.

20. print a list of the 20 biggest files you have (hint: sort, du, find)
21. Make an alias dos2unix that gets rid of ^M$ - very useful for those who still use Windows at home.
22. Make a shell script that given two programs executes them on given input files and reports any differences in the files (hint: man diff) this simulates a situation where you want to compare your program against the course's given solution. Very useful!
23. Write a shell script that solves the Hanoi Towers problem.

5.4.1 Answers

1. *locate fortune*. You probably will get many answers. The required file is /usr/games/fortune .
2. The PATH variable contains the list of directories where UNIX looks for executables. If the program you try to execute is not in any of the directories listed in the PATH variable, you get the *command not found* error. You can add the desired directory to your PATH or execute the full-name of the file: /usr/games/fortune.
3. *setenv PATH ${PATH}:/usr/games/*
4. fortune > bestFortunes; fortune >> bestFortunes; fortune >> bestFortunes
5. ln bestFortunes hardL
   ln -s bestFortunes softL
6. Even if bestFortunes is deleted hardL still exist, hence *cat hardL* works.
   On the other hand, softL is pointing to an non-existing file, hence *cat softL* will fail.
7. A hard link is another name (a synonym) to the same file. Hence any modification to bestFortunes, actually modifies hardL, and vice-versa. No new data is copied/created in the disk, hence the hard link does not occupy any space[3]. On the other hand, *cp* does copy the data, hence occupying disk space. Modifying the bestFortunes file, does not affect the copy.
8. ls | wc -l
   ls -a | wc -l # also hidden files/directories.
9. Note that although starting with '#', the first line (#!/bin/tcsh -f) is not a comment but mandatory.Do not forget to chmod 755 <file> to be able to execute it.

   ```
   #!/usr/bin/tcsh
   echo $2 $1
   ```

10. Note that although starting with '#', the first line (#!/bin/tcsh -f) is not a comment but mandatory. Do not forget to chmod 755 <file> to be able to execute it.

    ```
    #!/usr/bin/tcsh -f
    foreach file ( $1 )
       cp ${file} ${file}.bak
    end
    ```

[3]Actually a few bytes in the are used with each new hard-link, unrelated to the file size.

The tags are necessary, to impede the shell from expanding it.

11. grep CLASSPATH * .??* # .??*: check also the hidden files.
The -*i* makes grep treat upper case and lower case chars as the same, therefore also classpath, cLaSsPaTh, ... will be matched.

12. This is a very useful script in order to erase all those annoying core files.

```
#!/usr/bin/tcsh -f
find -name core -exec rm -i {} \;
```

The \; at the end is part of the find command, it tells it when the -exec command is finished.

13. This is a very useful script to find which .h file you need to #include for a certain C function.

```
#!/usr/bin/tcsh -f
find -name '*.h' -exec grep -H $1 {} \;
```

The -H tells grep to print the file name. The \; at the end is part of the find command, it tells it when the -exec command is finished

14. killall -9 firefox, or: kill ‘ps ux | grep mozilla-firefox | awk {print $2}‘

Chapter 6

Configuration Files and User Profile

6.1 Configuration Files

The configuration of various programs is stored in configuration files/directories[1]. Here are some of the most important configuration files:

`~/.cshrc` contains the initial script run by `tcsh` shell, before you can type any command. You can define your (user-specific) `PATH`, `CLASSPATH`, `PRINTER`, shell behavior, aliases, ...

`~/.xsession` is run when you login graphically. It determines which graphical environment you will have by default (xfce, ctwm, kde, gnome, fwm, enlightenment, ...). Note that on the 64 bit machines, the files is ~/.xsession64.

`~/.aliases` is read by `.cshrc` and contains alias definitions

`~/.rhosts` contains a list of hosts (computers) from which you can initiate `rsh` or `xon` commands, meaning that if you have, for example, `river.cs.huji.ac.il` listed there you can `rsh` or `xon` to any machine without having to enter password as long as you do it from river. When you initiate rsh command from host A to host B (on machine A you type: `rsh B <command>`)

1. A sends request to computer B to execute <command>,
2. B checks whether you have A listed in `~/.rhosts`
3. if yes, B lets you execute the command

`~/.classpath` contains your default `CLASSPATH` environment variable definition (read by .cshrc)

`~/.Xdefaults` (`~/.xrdb` on old systems) contains the X-Resources, some X-Windows System application can have configure information in this file (i.e. color settings, etc).

When all else fails, and you completely messed up you default configuration files, you may use the **reinstall** command to revert to the files as given to you when your account was open.

[1] These files/directories are usually named “dot-files”, since their name begins with a ’.’

6.2 Exercises

1. Enlarge the history size used by the shell.

Chapter 7

The Window Manager

7.1 What is X and what is a window manager?

X, or the X window system, allows you to run different graphical applications in different windows. Each window (or group of windows) belongs to an X application - a process. For example, your Iceweasel window (or windows if you opened more) belongs to the iceweasel process.

Just like our shell is a special kind of process, it is a Command Line Interpreter (CLI): it allows us to interact with UNIX using the keyboard and screen. By typing commands in the shell we may run different textual applications, and control them.

In a similar fashion, a window manager, as the name implies, is a special process that manages windows, i.e. it gives you tools to manipulate windows: drag them around, minimize them, resize them, etc.

Just like our shell allows humans to manipulate jobs, here we are provided with a Graphical User Interface (GUI) to interact with UNIX. We use the mouse and keyboard and have a pretty graphical screen, instead of just a text based shell.

There are many different window managers, and each window manager has it's advantages and disadvantages. We will present the default window manager we use here: gnome, but you may use any window manager that is installed.

Since we work in a Graphical Environment we need a special **textual terminal emulation program**. Either one of these: **xterm** or **gnome-terminal** can be used.[1]

In parallel a **tcsh** program is run, its input and output are **redirected** to the terminal emulator.

Now we can **use the xterm text-terminal emulator to textually communicate with the shell**.

7.2 The X session

If we return to the log on process you may remember we are presented with a nice login screen. After you log in, your ~/.xsession64 file is run.

You may look at ~/.xsession64 to see an example of what is run.

All the startup programs in ~/.xsession64 are run in the background. Otherwise our ~/.xsession64 would get stuck forever waiting for e.g. xclock (or something similar) to exit in order to continue.

The final program to be run in the ~/.xsession64 is our favorite window manager. This is NOT run in the background, because the moment your ~/.xsession64 is finished running, you are kicked out of X and your X session comes to an end. So

[1] We will see how to open **textual terminal emulation programs** when we get to your window manager.

we will want to run our window manager in the foreground in order to terminate our X session only once it has terminated (i.e. we chose exit).

The windows manager might also have some scripts and programs that it runs which are configured elsewhere (depending on the window manager).

7.3 Some basic X operations

Here are some basic operation as they are configured in the default environment.

Cut & Paste under UNIX:

This is unrelated to your window manager, copy and paste in UNIX is done with mouse in the following fashion:

- **copy**: select the text. i.e. left mouse button + drag
- **paste**: press the middle mouse button (or wheel) where you want to paste. Another method for paste is **shift+insert**.

Some programs use a different method which is more like the one from Windows, that is, ctrl+c and ctrl+v. Depending on the program this might and might not work with the standard mouse copy/paste.

Hebrew

To switch between Hebrew and English, press **both shift** at the same time, or use the mouse to press the flag on the panel. This key configuration can be changed, however we strongly encourage not to modify it (especially not to alt + shift), as this might interfere with other key bindings.

Window moving resizing

To move windows either drag it (left mouse button) through it's title bar (like in windows and mac) or press alt and left click anywhere on the window.
To resize a window, either drag the edges of the window or press alt and right click anywhere on the window.
Moving and resizing is also possible via the keyboard using Alt + F7 and Alt + F8.

zap

Zapping the X server means unconditionally, ungracefully, forcefully, brutally killing it. You will be instantly logged out, no program will have the chance to save anything, and some configuration files might get corrupted (so program will have a hard time launching again next time). It is only here to warn you not to use it. To zap the X server press: ctrl+alt+backspace (but don't press it).

7.4 Exercises

1. Write a shell command that starts xterm with a dark blue background and light green foreground. Hint: man xterm.
2. Add some lines in .Xdefaults so xterm will start by default in blue and green.

Chapter 8

Some Useful Applications

8.1 Emacs

Is of course the most useful one, helping you to program in almost any computer language. We already talked about it in section 3 on page 33. To learn keyboard bindings use the emacs-reference card.

8.2 LYX - Scientific Word Processor

> "The problem with WYSIWYG is that what you see is all you've got."
>
> - Brian Kernighan

Lyx is a WYSIWYM (What You See Is What You Mean) Word Processor. You specify the logic of the document, the program typesets it using sophisticated automated algorithms developed by Donald Knuth in a system called TEX. This document was written using Lyx.

- Logical Outline,
- Labels,
- References,
- Index,
- Math Formulae,
- Good Help

try:

```
lyx [<filename>] &
```

8.3 Dia - Diagram Drawing

A diagram/schema drawing utility.

- In order to incorporate dia diagrams into lyx you need to export them to "eps" files.

try:

```
dia &
```

8.4 evince

A program for displaying (and printing) pdfs and postscript files. Also known as Document viewer.

8.5 gv

A program to watch and print postscript files.

8.6 acroread

Program for displaying (and printing) pdf files. Sometime, it's useful to convert pdf files to postscript and for that you can use: pdf2ps.

8.7 DDD - Data Display Debugger

Enables you to visualize complex data structures in your running program.

➥ Works on C, C++, Java programs.

try:

```
ddd <executable-name> &
```

8.8 GDB - The GNU Debugger

This is the engine behind DDD. A powerfull c,c++ debugger with a textual interface. Can be used to trace programs in real time, or to examine core files from crashed programes.

```
gdb <executable-name> &
```

8.9 strace

Useful to track what system calls the program does.

8.10 Gimp

Photoshop like program for image processing:

```
gimp &
```

8.11 Offices

openoffice run it by *ooffice*

abiword a word processor

gnumeric a spreadsheet

8.12 Browsers

If you don't like iceweasel (or firefox), you can use one of the other borwers isntalled:

- lynx - a textual interfaced web browser
- epiphany - a very simple browser from GNOME
- chromium-browser - a web broweser from google

8.13 Empathy

An instant messaging client capable of connecting to: ICQ, AIM, MSN, Yahoo!, Google talk, XMPP, IRC, and more all at the same time.

8.14 xevil

A nice application to pass the time instead of doing your exerices. To see other such applications, take a look at /usr/games/

8.15 vlc

A media player (music, video, etc.)

8.16 ... + approx. 2000 different programs and utilities

Chapter 9

Troubleshooting

9.1 Restoring Files from Backup (Snapshot)

Your home directory is backup-ed several times a day. These backups are called snapshots.

If you deleted or otherwise ruined one of your files by mistake, you can recover it from the last valid snapshot. Note, however, that the backup contains files as they were at the time of the last snapshot. If you happen to have modified a file after the last snapshot, all your changes will be lost. You can easily lose several hours or even days of work! Please be careful when you are deleting files, i.e.*: do not count on the snapshot!!!*.

➥ `snapshot`:
  Type 'help' in snapshot for further information.

9.2 Problems in Login

Sometimes, when you login to a computer, after writing your username and password, the computer will throw you out, back to the login window. You will not get any indication on what went wrong. The two most frequent reasons for this "rude" behavior are:

1. Bad X Windows startup settings

   ➥ When you logon in graphical mode (X), many files are read and executed to set up your windowing environment. If any of these files has a severe problem, your environment will not be set correctly, and the environment will throw you out.

2. You exceeded your disk quota.

   ➥ Some programs (like KDE environment) write/modify files in your home directory when you log in. If you have no space (quota) left, this programs will fail in writing. As a result, you will be thrown out, without any visible error message.

Fortunately for us, there are ways of logging-in in the so called "failsafe" mode (i.e. a mode where your X settings files are not read and nothing is written to your home directory).

Depending on the computer, there are several ways you can enter the "failsafe mode". One of the following should work:

- In the login window, go to the SESSION menu (under the username/password prompt) and select FAILSAFE TERMINAL session. Then enter your login and password as usual.
- Enter your username when prompted (press OK button or ENTER), then enter your password but instead of pressing ENTER, press F1.
- When prompted to choose a program to run (Xfce, KDE, CTWM, etc) choose FAILSAFE/XTERM. This is after you've entered your username + password. This only worked on the old system.

Once you entered the failsafe mode, you will get a simple xterm (shell) without any other window. To fix the problem:

- Check your disk quota/usage using the nquota and du commands. See 9.3
- Check Xsession error messages by reading your ~/.xsession-errors file. This file contains all error messages generated during your X login. If you see a line like: *cannot find <program>*, check your .xsession or .cshrc file to see where these programs are called and try to solve the problem. You can also reinstall all or a part of the default configuration, using the reinstall command. See 9.4

9.3 Freeing Disk Space

You can check how much disk space is available to you by using the *nquota* command. Here is an example of its output:

Filesystem	usage	quota	files	quota
/cs/stud	20532	40960	988	40960

This means that you are using 20532 KB (~20 MB) out of 40960 KB (~40 MB) you are allowed to use. The 4th and 5th column refer to the number of files you have, no matter what their total size is: You have 988 files, and can have a maximum of 40960 files).

There is a script that will clean up some extra files that curtain programs create but you don't really need (all sort of cache files). To get rid of those you should run the "cleanup" script and it will clean up those files.

You can use the 'du' command to check which directories are wasting the most of your space:

```
cd          #To switch to your home directory

du -s *   .??* | sort -n
```

or

```
dua   # run ''which dua'' to see what it does
```

This will show you how much each file/directory is using, sorted from smallest to largest

After you have located a directory that uses a lot of space, *cd* to that directory, and run above *du* again. Continue this procedure, recursively until you have located the largest problematic files. Finally delete them using the *rm* command.

Notes:

- ➥ A <file>.class can be created from a <file>.java, so after submitting your exercise, you can safely delete the .class files.[1]
- ➥ The same is true for C and other programming languages: For instance, .o and .a files, are created by compiling .c and .h files. You can safely those after submitting your exercise.
- ➥ Mozilla-Firefox/opera/konqueror and other internet browsers, store old files you have been browsing through in your home directory. Go to the Options/Preferences/Setting on your browser, and un-check (or set 0) the *disk cache* option.
- ➥ Don't forget, you can move large files that don't need regular backup in to your ~/.+/ directory and free some space in your home directory.
- ➥ Check `http://wiki/wiki/Disk_space#Disk_space_shortage` for more info on disk shortage problems and fixes.

9.4 Restoring the Original Environment

You can restore part or all of your original environment by following these simple steps:

1. **Logout**, or if unable to, press **CTRL-ALT-BACKSPACE** (not delete).
2. Login into "**Failsafe**" mode (see 9.2).
3. In the shell window, type "`reinstall`" at the prompt, press **ENTER** and follow the instructions.
4. Type "**exit**".
5. Try to login as usual.
6. If your problems persist, call system.

[1] You may prefer to wait until you've got your grade, before deleting... Ask your intro2cs TAs, for more info.

Index