Grundpraktikum Bioinformatik

SS 2016

November 26, 2018

AG Ullmann

- **Jeder Praktikumstag ist ein Chapter.**
- **Vorsicht!!! Der Inhalt dieser Praktiumsanleitung kann sich bis zu einem Tag vor dem Praktikumstermin ändern! Bitte kapitelweise ausdrucken.**
- `l`$\neq$`1`**, d.h. ein kleines L ist keine Eins**
- `O`$\neq$`0`**, d.h. ein großes o ist keine Null**
- `[CTRL]=[STRG]`
- **`‘`$\neq$`’`, d.h. es gibt verschiedene Anführungszeichen**

Bitte beachten Sie

Wenn Sie den Raum verlassen:

Zur Mittagspause sollten sie den Bildschirmschoner starten. Dafür drücken Sie die rechte Maustaste auf dem Desktop-Hintergrund und wählen ”Lock Screen”. Nach der Mittagspause müssen Sie dann Ihr Passwort eingeben, um weiter arbeiten zu können. Dafür kann in der Zwischenzeit niemand Ihren Rechner verwenden.

Nach Abschluss des Praktikums loggen Sie sich bitte aus. Aber schalten Sie die Rechner nicht aus, sie werden oft zum Rechnen über Nacht verwendet.

Wie Sie dieses Praktikum erfolgreich abschließen:

- Der Stoff der Vorlesung ist Voraussetzung für das Arbeiten im Praktikum. Gehen Sie also zur Vorlesung oder sehen Sie sich also zumindest die Folien an!
- Arbeiten Sie im Verzeichnis ”Praktikum/Tag_X” (X=1,2,3,4,5). Das Erstellen dieser Verzeichnisse wird eine der Aufgaben des ersten Praktikumstages sein, also keine Panik wenn nicht gleich klar ist, was Sie hier tun sollen.
- Das Praktikum hat z. T. den Charakter einer Übung. Deshalb ist das Protokoll in eine etwas unüblichen Form abzugeben. Am Ende jeder Praktikumsanleitung werden Fragen gestellt. Beantworten Sie diese Fragen in einem kurzen Protokoll (Ein Protokoll pro Praktikumstag). Sie können dieses Protokoll z. B. mit LibreOffice (Befehl ”`libreoffice`”) im CIP-Pool schreiben.
- Arbeiten Sie in Zweier-Gruppen und geben Sie ein Protokoll pro Gruppe ab. Schreiben Sie beide Namen und Benutzerkennungen auf das Protokoll.
- Wann Sie die Mittagspause machen, ist Ihnen freigestellt.

Protokoll – Abgabe und Korrektur

- Geben Sie die ausgedruckten Protokolle spätestens eine Woche nach dem Praktikumstag ab (z.B. am Dienstag in der Vorlesung oder am Mittwoch im Praktikumsraum).
- Diese Protokolle werden in der folgenden Woche korrigiert. Für jedes richtige Protokoll gibt es einen Punkt. Sollte ein Protokoll Fehler enthalten, haben Sie **einmal** die Möglichkeit, das Protokoll zu verbessern. **Geben Sie die Verbessung zum Protokoll zusammen mit dem vorherigen fehlerhaften Protokoll eine Woche nach Rückgabe ab**.
- Falls Sie:
  - Protokolle nicht innerhalb einer Woche abgeben,
  - das korrigierte Protokoll nicht innerhalb einer Woche abgeben, bzw.
  - das korrigierte Protokoll fehlerhaft ist,

haben Sie die Möglichkeit, Ihr Wissen zu diesem Thema in einem **Kolloquium bei Prof. Ullmann** unter Beweis zu stellen. Falls Sie das Kolloquium nicht bestehen, können Sie es **einmal wiederholen**. Falls Sie es auch ein zweites Mal nicht bestehen, können Sie das Praktikum **im darauffolgenden Jahr** wiederholen.

- Um das Praktikum zu bestehen und zur Prüfung zugelassen zu werden, müssen alle Punkte erreicht werden.
- Sie können auf dieser Seite [1] nachprüfen, ob Sie auf ein Protokoll einen Punkt bekommen haben.
- Die Protokolle werden zu Beginn des darauf folgenden Praktikumstages besprochen.

Wie soll das Protokoll aussehen?

- Schreiben Sie das Protokoll auf Deutsch.
- Schreiben Sie das Protokoll z. B. mit LibreOffice (Kommando libreoffice). Dieses Programm ist MS Office ähnlich.
- Kopieren Sie keine Protokolle oder Abbildungen von Kommilitonen. Das kann ersthafte rechtliche Konsequenzen haben (Plagiat!).
- Beantworten Sie die Fragen/Aufgaben (Tasks) in kurzen aussagekräftigen Sätzen. Geben Sie die Nummern der Fragen/Aufgaben an! Kopieren Sie die Fragen mit in das Protokoll!
- Schreiben Sie aussagekräftige Legenden zu Abbildungen und Tabellen. Vergessen Sie bei Diagrammen die Achsenbeschriftung und ggf. die Einheiten nicht!

[1] http://www.bisb.uni-bayreuth.de/People/ullmann/praktikum/points.html

Chapter 1

UNIX Tutorial for Beginners

This part is a short introduction to Linux (and other Unix like systems). [1]

1.1 Typographical conventions

In what follows, we shall use the following typographical conventions:

- Characters written in `typewriter font` are commands to be typed into the computer as they stand.
- Characters written in *italic font* indicate non-specific file or directory names.
- Words inserted within square brackets [Ctrl] indicate keys to be pressed.
- The string "`prompt:`" is called the prompt. The look of this string is dependent on the system you are working with or actually on your (default) settings. For our system that prompt contains your current working directory and a ">" sign (like `bpbi0815/unixstuff>`). Sometimes it is just "%" or ">". Whatever you prompt might be, don't type the prompt with your commands!

So, for example,

`prompt:` `ls` *anydirectory* [Enter]

means "at the UNIX prompt "`prompt:`", type "`ls`" followed by the name of some directory, then press the key marked Enter".

Don't forget to press the [Enter] key: commands are not sent to the computer until this is done.

Note: UNIX is case-sensitve, so LS is not the same as ls. The same applies to filenames, so myfile.txt, MyFile.txt and MYFILE.TXT are three seperate files.

[1] originally taken from http://www.ee.surrey.ac.uk/Teaching/Unix/ but largely extended and modified

1.2 UNIX Introduction

This session concerns UNIX, which is a common operating system. By operating system, we mean the suite of programs which make the computer work. UNIX is used by workstations and servers within the university.

On X terminals and the workstations, X Windows provide a graphical interface between the user and UNIX. However, knowledge of UNIX is required for operations which aren't covered by a graphical program, or for when there is no X windows system, for example, in a ssh session.

1.2.1 The UNIX operating system

The UNIX operating system is made up of three parts; the kernel, the shell and the programs.

The kernel

The kernel of UNIX is the hub of the operating system: it allocates time and memory to programs and handles the filestore and communications in response to system calls.

As an illustration of the way that the shell and the kernel work together, suppose a user types **`rm myfile`** (which has the effect of removing the file **`myfile`**). The shell searches the filestore for the file containing the program **`rm`**, and then requests the kernel, through system calls, to execute the program **`rm`** on **`myfile`**. When the process **`rm myfile`** has finished running, the shell then returns the UNIX prompt `prompt:` to the user, indicating that it is waiting for further commands.

The shell

The shell acts as an interface between the user and the kernel. When a user logs in, the login program checks the username and password, and then starts another program called the shell. The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt.

The user can customize his/her own shell, and users can use different shells on the same machine. An comfortable shell is the tcsh shell which has certain features to help the user inputting commands:

- **Filename Tab-Completion:** By typing part of the name of a command, filename or directory and pressing the [Tab] key, the tcsh shell will complete the rest of the name automatically. If the shell finds more than one name beginning with those letters you have typed, it will beep, prompting you to type a few more letters before pressing the tab key again.

- **History:** The shell keeps a list of the commands you have typed in. If you need to repeat a command, use the cursor keys to scroll up and down the list or type history for a list of previous commands.

- **Mouse Buffer:** Copying and pasting text in Linux between different text windows can be easy. You just have to:

  1. mark the text which you want to copy with the left mouse button
  2. click with the left mouse button to where you want to paste the text
  3. press the middle mouse button to paste the text

  If you want to copy from PDF files (such as your script), you can for instance use the program `okular` and use the [Selection] menu:

  1. Click on Selection in the menu (it should become framed with a somewhat darker background)
  2. mark the part which you want to copy with the left mouse button (i. e. put a frame)
  3. click with the left mouse button on "Copy to Clipboard"
  4. click with the left mouse button to where you want to paste
  5. press the middle mouse button to paste the text

1.2.2 Files and Processes

Everything in UNIX is either a file or a process.

A process is an executing program identified by a unique PID (process identifier).

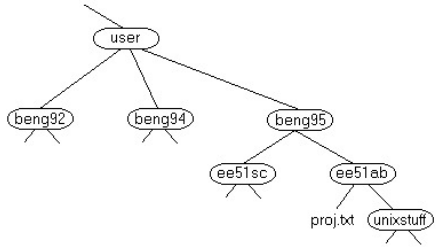
A file is a collection of data. They are created by users using text editors, running compilers etc.

Examples of files:

- a document (report, essay etc.)
- the text of a program written in some high-level programming language
- instructions comprehensible directly to the machine and incomprehensible to a casual user, for example, a collection of binary digits (an executable or binary file)
- a directory, containing information about its contents, which may be a mixture of other directories (subdirectories) and ordinary files.

1.2.3 The Directory Structure

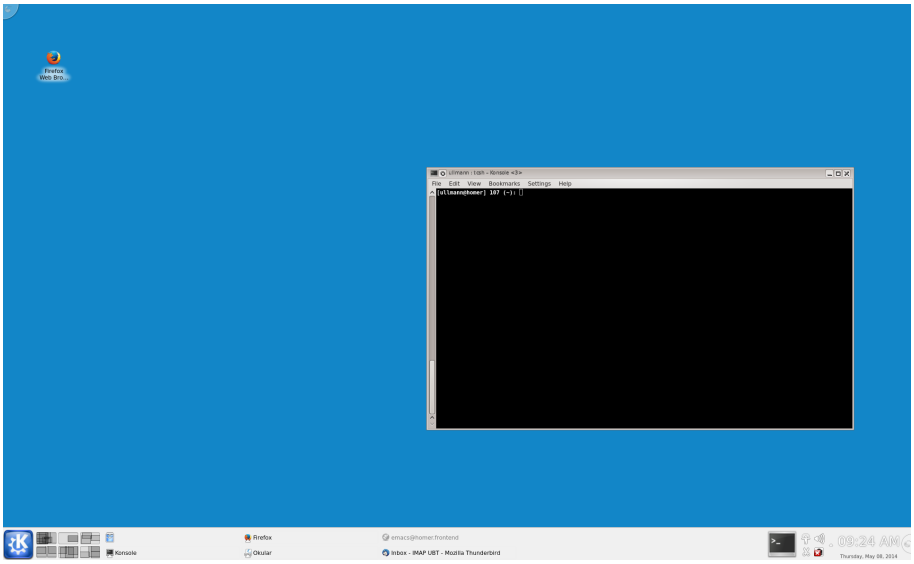
All the files are grouped together in the directory structure. The file-system is arranged in a hierarchical structure, like an inverted tree. The top of the hierarchy is traditionally called root.



In the diagram above, we see that the directory ee51ab contains the subdirectory unixstuff and a file proj.txt

1.2.4 Graphical Desktop Environment

All modern Unix systems come with a graphical desktop environment which help you to handle processes and software. There are several different graphical desktop environments and there are a lot of possibilities to configure them. One popular desktop environments is KDE, which can look like this:



In the left lower corner you have a start button (similar to Windows), there are icons on the desktop (here Firefox), and there is a terminal window.

A terminal allows you to type commands in the command line and you can access the files and directories on you system through this terminal. Terminals can be also run without a graphical interface or also on remote computers, for instance on large compute clusters in computing centers. Knowing how to use a the command line is essential for working efficiently in computational science. Once you got used to it, you wonder how you could have ever used a computer without a command line interface. The major goal of today's practical is to introduce you how to use the command line.

1.2.5 Web Browser

In the days of the internet, an efficient webbrowser and easy-to-use webbrowser is essential. We recommend to use firefox, because it run on Linux as well as on Windows and Macs. Thus, once you know how to use it, you can use it efficiently on different operation systems. In case you are not familiar with firefox, it is worth to take ten minutes to try a few things:

- With [CTRL]+[+] and [CTRL]+[-] you can increase and decrease the size of the content of the web page

- By right-clicking on a link, you can choose to open the linked page either in a new tab or a new window; or also to save a file
- By clicking on the space just below the top frame, you can switch on and off a menu bar, a bookmarks bar etc.
- You can bookmark pages that are interesting to you.
- In `Preferences`, you can set you starting page (Home Page), how you want to handle download and many more things.

1.3 UNIX – Files and Directories

1.3.1 Listing files and directories

When you first login, you will get a desktop environment of Linux. In order to type your commands, you will need a terminal. Getting a terminal window is dependent on the type and version of the desktop manager you are using. What should normally work is clicking on the icon in the left lower corner, search for terminal and choose one of them (for instance `konsole`). Your current working directory is your home directory. Your home directory has the same name as your user-name, for example, `bpbi12`, and it is where your personal files and subdirectories are saved. In your terminal window, you should see something like

```
BI/bpbi01>
```

This is your prompt, which we will show as `prompt:` from now on. Do not type `prompt:`, when you enter your commands!!!! The prompt may change when you are working.

ls (list)
To find out what is in your home directory, type

```
prompt:  ls
```

The `ls` command (= short for list) lists the contents of your current working directory.

There may be no files visible in your home directory, in which case, the UNIX prompt will be returned. Alternatively, there may already be some files inserted by the System Administrator when your account was created.

`ls` does not, in fact, cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot (.) Files beginning with a dot (.) are known as hidden files and usually contain important program configuration information. They are hidden because you should not change them unless you are very familiar with UNIX!!!

To list all files in your home directory including those, whose names begin with a dot, type

```
prompt:  ls -a
```

`ls` is an example of a command which can take options: `-a` is an example of an option. The options change the behavior of the command. There are online manual pages that tell you which options a particular command can take, and how each option modifies the behavior of the command (See later in this tutorial). In most unix programs, options start with a '-' sign, like here for instance '`-a`'.

1.3.2 Making Directories

We will now make a subdirectory in your home directory to hold the files you will be creating and using in the course of this tutorial. To make a subdirectory called unixstuff in your current

working directory type

```
prompt:  mkdir unixstuff
```

To see the directory you have just created, type

```
prompt:  ls
```

1.3.3 Changing to a different directory

cd (change directory)

The command `cd` *directory* means change the current working directory to 'directory'. The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree.

To change to the directory you have just made, type

```
prompt:  cd unixstuff
```

Type `ls` to see the contents (which should be empty).

Exercise 1a: Make another directory inside the **unixstuff** directory called **backups**.
By typing

```
prompt:  pwd
```

you can always see in which directory you are at the moment.

1.3.4 The directories . and ..

Still in the unixstuff directory, type

```
prompt:  ls -a
```

As you can see, in the **unixstuff** directory (and in all other directories), there are two special directories called (.) and (..)

In UNIX, (.) means the current directory, so typing

```
prompt:  cd .
```

means stay where you are (in the **unixstuff** directory).

This may not seem very useful at first, but using (.) as the name of the current directory will save a lot of typing, as we shall see later in the tutorial.

(..) means the parent of the current directory (one level up), so typing

```
prompt:  cd ..
```

will take you one directory up the hierarchy (back to your home directory). Try it now.

Typing **cd** with no argument always returns you to your home directory. This feature is very useful if you are lost in the file system.

1.3.5 Pathnames

Understanding pathnames

First type `cd` to get back to your home-directory, then type

```
prompt:  ls unixstuff
```

to list the contents of your unixstuff directory.

Now type

```
prompt:  ls backups
```

You will get a message like this

```
backups:  No such file or directory
```

The reason is, **backups** is not in your current working directory. To use a command on a file (or directory) not in the current working directory (the directory you are currently in), you must either **cd** to the correct directory, or specify its full pathname. To list the contents of your backups directory, you must type

```
prompt:  ls unixstuff/backups
```

~ (your home directory)

Home directories can also be referred to by the tilde ~ character. It can be used to specify paths starting at your home directory. So typing

```
prompt:  ls ~/unixstuff
```

will list the contents of your unixstuff directory, no matter where you currently are in the file system.

What do you think `ls ~` would list?

What do you think `ls ~/..` would list?

1.3.6 Efficient working with the mouse and coursers

Now you have typed already some commands. As said above, the shell keeps a history of your commands. In order to save typing work, you can access your previous commands. For

instance, if you need to repeat a command, use the cursor keys [up] and [down] to go up and down the list or type history for a list of previous commands. You may modify your commands by using the cursor keys [left] and [right] to position your cursor, [Tab] or [del] to delete parts, and the other keys to type things. Try it out!

Another comfortable help in the tcsh shell (which is what you should be using) is the so-called Tab-Completion. By typing part of the name of a command, filename or directory name and pressing the [Tab] key, the tcsh shell will complete the rest of the name automatically. If the shell finds more than one name beginning with those letters you have typed, it will beep, prompting you to type a few more letters before pressing the tab key again. Try it, type:

```
prompt:  ls unix[Tab]
```

The shell should complete the word to `unixstuff`.

Also the mouse buffer is a useful tool in most unix systems. With its help, copying and pasting text in Linux between different text windows or even within the same text window is easy. You just have to:

1. mark the text which you want to copy with the left mouse button
2. click with the left mouse button to where you want to paste the text
3. press the middle mouse button to paste the text

Try the mouse buffer for the last command you where typing.
Try to use these tools are you go along in this tutorial.

1.3.7 Summary

`ls`	list files and directories
`ls -a`	list all files and directories
`mkdir`	make a directory
`cd directory`	change to named directory
`cd`	change to home-directory
`cd ~`	change to home-directory
`cd ..`	change to parent directory
`pwd`	display the path of the current directory

1.4 Copy, Move and other File Handlings

1.4.1 Copying Files — cp (copy)

`cp file1 file2` is the command which makes a copy of `file1` in the current working directory and calls it `file2`.

What we are going to do now, is to take a file stored in an open access area of the file system, and use the cp command to copy it to your unixstuff directory.

First, `cd` to your unixstuff directory.

```
prompt:  cd ~/unixstuff
```

Then at the UNIX prompt, type,

```
prompt:  cp /cip1/Kurse/BI/bpbi00/science.txt .
```

Do not forget the dot (.) at the end. Remember, in UNIX, the dot means the current directory.

The above command means copy the file science.txt to the current directory, keeping the name the same.

The above directory is an area to which everyone in the CIP pool has read and copy access. If you are from outside the CIP pool, you can get a copy of the file here. [2] You can download the file through your webbrowser and save the file in your unixstuff directory.

Alternatively, you can also use the command line program `wget` to download the file:

```
prompt:  wget http://www.bisb.uni-bayreuth.de/People/ullmann/pract/science.txt
```

Exercise 2a: Create a backup of your science.txt file by copying it to a file called "`science.bak`".

1.4.2 Moving files — mv (move)

`mv file1 file2` moves (or renames) `file1` to `file2`

To move a file from one place to another, use the `mv` command. This command has the effect of moving rather than copying the file, so you end up with only one file rather than two.

It can also be used to rename a file, by moving the file to the same directory, but giving it a different name.

Do not use `rename` unless you know how to use it!

We are now going to move the file science.bak to your backup directory.

First, change directories to your unixstuff directory. Then, inside the **unixstuff** directory, type

```
prompt:  mv science.bak backups
```

[2] http://www.bisb.uni-bayreuth.de/People/ullmann/pract/science.txt

Type `ls` and `ls backups` to see if it has worked.

1.4.3 Removing files and directories — rm (remove), rmdir (remove directory)

To delete (remove) a file, use the `rm` command. As an example, we are going to create a copy of the science.txt file then delete it.

Inside your unixstuff directory, type

```
prompt:  cp science.txt tempfile.txt
prompt:  ls  (to check if it has created the file)
prompt:  rm tempfile.txt
prompt:  ls  (to check if it has deleted the file)
```

You can use the `rmdir` command to remove a directory (make sure it is empty first). Try to remove the **backups** directory. You will not be able to since UNIX will not let you remove a non-empty directory.

Exercise 2b: Create a directory called tempstuff using `mkdir`, then remove it using the `rmdir` command.

1.4.4 Removing files and directories

clear (clear screen)

Before you start the next section, you may like to clear the terminal window of the previous commands so the output of the following commands can be clearly understood.

At the prompt, type

```
prompt:  clear
```

This will clear all text and leave you with the % prompt at the top of the window.

cat (concatenate)

The command `cat` can be used to display the contents of a file on the screen. Type:

```
prompt:  cat science.txt
```

As you can see, the file is longer than than the size of the window, so it scrolls past making it unreadable.

more

The command `more` writes the contents of a file onto the screen a page at a time. Type

```
prompt:  more science.txt
```

Press the [space-bar] if you want to see another page, type [q] if you want to quit reading. As you can see, `more` is used in preference to `cat` for long files.

less

The command `less` is similar to `more`, but has some additional features. Thus, less is more. ;-)
(See below!)

head

The `head` command writes the first ten lines of a file to the screen.
First clear the screen then type

```
prompt:  head science.txt
```

Then type

```
prompt:  head -8 science.txt
```

What difference did the -8 do to the head command?

tail

The tail command writes the last ten lines of a file to the screen.
Clear the screen and type

```
prompt:  tail science.txt
```

How can you view the last 15 lines of the file?

paste

merges lines of files.

```
prompt:  paste science.txt science.txt
```

1.4.5 Searching the contents of a file

Simple searching using less

Using `less`, you can search though a text file for a keyword (pattern or regular expressions). For example, to search through `science.txt` for the word 'science', type

```
prompt:  less science.txt
```

then, still in `less` (i.e. don't press [q] to quit), type a forward slash [/] followed by the word to search

```
/science
```

As you can see, `less` finds and highlights the keyword. Type [n] to search for the next occurrence of the word.

grep (global search for a regular expression and print out matched lines)

`grep` is one of many standard UNIX utilities. It searches files for specified words or patterns. First `clear` the screen, then type

```
prompt:  grep science science.txt
```

As you can see, `grep` has printed out each line containg the word `science`.

Or has it????

Try typing

```
prompt:  grep Science science.txt
```

The `grep` command is case sensitive; it distinguishes between `Science` and `science`.

To ignore upper/lower case distinctions, use the `-i` option, i.e. type

```
prompt:  grep -i science science.txt
```

To search for a phrase or pattern, you must enclose it in single quotes (the apostrophe symbol). For example to search for spinning top, type

```
prompt:  grep -i 'spinning top' science.txt
```

Some of the other options of grep are:

`-v` display those lines that do NOT match
`-n` precede each maching line with the line number
`-c` print only the total count of matched lines

Try some of them and see the different results. You can use more than one option at a time, for example, the number of lines without the words `science` or `Science` is

```
prompt:  grep -ivc science science.txt
```

wc (word count)

A handy little utility is the **wc** command, short for word count. To do a word count on `science.txt`, type

```
prompt:  wc -w science.txt
```

To find out how many lines the file has, type

```
prompt:  wc -l science.txt
```

1.4.6 Dealing with a PDB file

Protein structures are often save in PDB-files, where PDB stands for Protein Data Base. You will here about these files later in the course. These files are often used for calculations and need to be manipulated and analyzed in on way or another. A PDB-file contains the information on the xyz-coordinates of the protein atoms, but also much more information, such as the name of the molecule, the sequence, the publication associated with the structures etc. A typical pdb file looks like this:

```
HEADER    ION TRANSPORT                           13-JUN-02   1M0L
TITLE     BACTERIORHODOPSIN/LIPID COMPLEX AT 1.47 A RESOLUTION
COMPND    MOL_ID: 1;
COMPND   2 MOLECULE: BACTERIORHODOPSIN;
....
JRNL        AUTH   B.SCHOBERT,J.CUPP-VICKERY,V.HORNAK,S.SMITH,J.LANYI
JRNL        TITL   CRYSTALLOGRAPHIC STRUCTURE OF THE K INTERMEDIATE OF
JRNL        TITL 2 BACTERIORHODOPSIN: CONSERVATION OF FREE ENERGY AFTER
JRNL        TITL 3 PHOTOISOMERIZATION OF THE RETINAL.
JRNL        REF    J.MOL.BIOL.                   V. 321   715 2002
JRNL        REFN                   ISSN 0022-2836
JRNL        PMID   12206785
JRNL        DOI    10.1016/S0022-2836(02)00681-2
REMARK   2
REMARK   2 RESOLUTION.    1.47 ANGSTROMS.
REMARK   3
....
SEQRES   1 A  262  MET LEU GLU LEU LEU PRO THR ALA VAL GLU GLY VAL SER
SEQRES   2 A  262  GLN ALA GLN ILE THR GLY ARG PRO GLU TRP ILE TRP LEU
SEQRES   3 A  262  ALA LEU GLY THR ALA LEU MET GLY LEU GLY THR LEU TYR
SEQRES   4 A  262  PHE LEU VAL LYS GLY MET GLY VAL SER ASP PRO ASP ALA
SEQRES   5 A  262  LYS LYS PHE TYR ALA ILE THR THR LEU VAL PRO ALA ILE
.....
ATOM      1  N   THR A   5      24.031  25.737 -13.776  1.00 61.46           N
ATOM      2  CA  THR A   5      23.434  25.735 -15.107  1.00 62.71           C
ATOM      3  C   THR A   5      24.050  26.801 -16.008  1.00 52.83           C
ATOM      4  O   THR A   5      25.230  27.143 -15.939  1.00 57.70           O
ATOM      5  CB  THR A   5      21.904  25.959 -15.048  1.00 72.95           C
ATOM      6  OG1 THR A   5      21.606  27.323 -14.739  1.00 65.89           O
ATOM      7  CG2 THR A   5      21.268  25.125 -13.939  1.00 79.47           C
ATOM      8  N   GLY A   6      23.188  27.328 -16.866  1.00 54.58           N
ATOM      9  CA  GLY A   6      23.573  28.387 -17.785  1.00 61.88           C
ATOM     10  C   GLY A   6      23.443  29.740 -17.109  1.00 57.17           C
ATOM     11  O   GLY A   6      23.744  30.785 -17.686  1.00 66.26           O
ATOM     12  N   ARG A   7      22.983  29.700 -15.858  1.00 38.09           N
ATOM     13  CA  ARG A   7      22.883  30.944 -15.098  1.00 33.50           C
ATOM     14  C   ARG A   7      24.150  31.066 -14.279  1.00 24.94           C
...........

ATOM   1718  CA  GLY A 231      15.552  30.886  32.019  1.00 46.38           C
ATOM   1719  C   GLY A 231      16.523  30.690  33.171  1.00 48.04           C
ATOM   1720  O   GLY A 231      17.627  30.201  32.976  1.00 43.33           O
TER    1721      GLY A 231
HETATM 1722  C1  LI1 A 601       4.259  63.398  -7.884  1.00 74.72           C
HETATM 1723  C2  LI1 A 601       4.424  64.911  -7.689  1.00 66.62           C
HETATM 1724  C3  LI1 A 601       5.774  65.371  -8.257  1.00 73.95           C
HETATM 1725  O1  LI1 A 601       3.174  63.197  -8.844  1.00 89.86           O
HETATM 1726  O2  LI1 A 601       4.306  65.241  -6.294  1.00 66.87           O
HETATM 1727  O3  LI1 A 601       5.535  65.490  -9.785  1.00 71.99           O
....
END
```

The lines starting with `ATOM` or `HETATOM` contain the information on the protein atoms and on the cofactor atoms, respectively (such as atom name, residue name, residue number, xyz-coorinates of the atoms etc.). An example PDB file, you can download directly from the PDB using `wget`.

```
prompt:  wget http://www.rcsb.org/pdb/files/1c3w.pdb
```

If you want now for instance to get all the lines that contain TRP (for tryptophan), you can type

```
prompt:  grep TRP 1c3w.pdb
```

You obtain a relatively long list and you may have to scroll to see the beginning of the list.

Exercise: Get a list of all lines that contain CA (for C_α of the aminoacid).

1.4.7 Summary

`cp file1 file2`	copy file1 and call it file2
`mv file1 file2`	move or rename file1 to file2
`rm file`	remove a file
`rmdir directory`	remove a directory
`cat file`	display a file
`more file`	display a file a page at a time
`head file`	display the first few lines of a file
`tail file`	display the last few lines of a file
`grep 'keyword' file`	search a file for keywords
`wc file`	count number of lines/words/characters in file

1.5 Redirecting and Piping

1.5.1 Redirection

Most processes initiated by UNIX commands write to the standard output (that is, they write to the terminal screen), and many take their input from the standard input (that is, they read from the keyboard). There is also the standard error, where processes write their error messages, by default, to the terminal screen.

We have already seen one use of the `cat` command to write the contents of a file to the screen.

Now type `cat` without specifing a file to read

```
prompt:  cat
```

Then type a few words on the keyboard and press the [Return] key.

Finally hold the [Ctrl] (or [STRG]) key down and press [d] (written as ^D for short) to end the input.

What has happened?

If you run the `cat` command without specifing a file to read, it reads the standard input (the keyboard), and on receiving the'end of file' (^D), copies it to the standard output (the screen).

In UNIX, we can redirect both the input and the output of commands.

1.5.2 Redirecting the Output

We use the > symbol to redirect the output of a command. For example, to create a file called `list1` containing a list of fruit, type

```
prompt:  cat > list1
```

Then type in the names of some fruit. Press [Return] after each one.

```
pear
banana
apple
^D ([CTRL]+[SHIFT]+d to stop)
```

What happens is the `cat` command reads the standard input (the keyboard) and the > redirects the output, which normally goes to the screen, into a file called list1

To read the contents of the file, type

```
prompt:  cat list1
```

Exercise 3a: Using the above method, create another file called `list2` containing the following fruit: orange, plum, mango, grapefruit. Read the contents of `list2`.

The form >> appends standard output to a file. So to add more items to the file `list1`, type

```
prompt:  cat >> list1
```

Then type in the names of more fruit

```
peach
grape
orange
^D ([CTRL]+[SHIFT]+d to stop)
```

To read the contents of the file, type

```
prompt:  cat list1
```

You should now have two files. One contains six fruit, the other contains four fruit. We will now use the `cat` command to join (concatenate) `list1` and `list2` into a new file called biglist. Type

```
prompt:  cat list1 list2 > biglist
```

What this command is doing is reading the contents of `list1` and `list2` in turn, then outputing the text to the file biglist.
To read the contents of the new file, type

```
prompt:  cat biglist
```

1.5.3 Redirecting the Input

We use the < symbol to redirect the input of a command.

The command `sort` alphabetically or numerically sorts a list. Type

```
prompt:  sort
```

Then type in the names of some vegetables. Press [Return] after each one.

```
carrot
beetroot
artichoke
^D ([CTRL]+[SHIFT]+d to stop)
```

The output will be

```
artichoke
beetroot
carrot
```

Using < you can redirect the input to come from a file rather than the keyboard. For example, to sort the list of fruit, type

```
prompt:  sort < biglist
```

and the sorted list will be output to the screen.

To output the sorted list to a file, type,

```
prompt:  sort < biglist > slist
```

Use `cat` to read the contents of the file `slist`

1.5.4 Pipes

To see which programs are available in the directory (ordered by time) type

```
prompt:  ls -t /usr/bin
```

The `-t` gives a temporal order of the files. One method to get a lexicographically sorted list of names is to type,

```
prompt:  ls -t /usr/bin > names.txt

prompt:  sort < names.txt
```

This is a bit slow and you have to remember to remove the temporary file called `names.txt` when you have finished. What you really want to do is connect the output of the `ls -t /usr/bin` command directly to the input of the `sort` command. This is exactly what pipes do. The symbol for a pipe is the vertical bar `|`. (On your keyboard, the pipe symbol look like two vertical bars on top of each other).

For example, typing

```
prompt:  ls -t /usr/bin | sort
```

will give the same result as above, but quicker and cleaner.

To find out how many files are in `/usr/bin`, type

```
prompt:  ls /usr/bin | wc -l
```

Exercise 3b: Use pipes to get all lines of `list1` and `list2` containing the letter 'p' and sort the result.

1.5.5 Again dealing with a PDB file

Now we want to use pipes and redirections to deal with PDB files. Suppose you want to get a list of all the tryptophan atoms. Before, you were typing

```
prompt:  grep TRP 1c3w.pdb
```

but the list that you were obtaining was very long. Now if you pipe this output into more, you can just go through the output step by step:

```
prompt:  grep TRP 1c3w.pdb | more
```

But if you only want to get the lines that contain coordinates, you can combine two grep commands with a pipe.

```
prompt:  grep TRP 1c3w.pdb | grep ATOM | more
```

If you want to know the numbers of tryptophanes in your structure, you should just count one atom. The C_α is a good choice since there is only one C_α in each aminoacids. Thus,

```
prompt:  grep TRP 1c3w.pdb | grep ATOM | grep CA | more
```

If you are too lazy to count the lines yourself, let `wc` do the job for you.

```
prompt:  grep TRP 1c3w.pdb | grep ATOM | grep CA | wc -l
```

Exercise: Get the charge of the apo-protein of 1c3w.pdb at pH=7. Remember that aspartate and glutamate are negatively charged, lysine and arginine are positively charged. Histidine has a 50:50 chance of being positively charged. A single command may not be enough. You may have to type five command line commands.

1.5.6 Summary

`command > file`	redirect standard output to a file
`command >> file`	append standard output to a file
`command < file`	redirect standard input from a file
`command1 | command2`	pipe the output of command1 to the input of command2
`cat file1 file2 > file0`	concatenate file1 and file2 to file0
`sort`	sort data
`who`	list users currently logged in

1.6 Wildcards and Help

1.6.1 The characters * and ?

The character * is called a wildcard, and will match against none or more character(s) in a file (or directory) name. For example, in your `unixstuff` directory, type

```
prompt:  ls list*
```

This will list all files in the current directory starting with `list...`.

Try typing

```
prompt:  ls *list
```

This will list all files in the current directory ending with `....list`.

The character `?` will match exactly one character. So `ls ?ouse` will match files like `house` and `mouse`, but not `grouse`. Try typing

```
prompt:  ls ?list
```

1.6.2 Filename conventions

We should note here that a directory is merely a special type of file. So the rules and conventions for naming files apply also to directories.

In naming files, characters with special meanings such as `/ * & % ,` should be avoided. Also, avoid using **spaces** within names. The safest way to name a file is to use only alphanumeric characters, that is, letters and numbers, together with _ (underscore) and `.` (dot).

File names conventionally start with a lower-case letter, and may end with a dot followed by a group of letters indicating the contents of the file. For example, all files consisting of C code may be named with the ending `.c`, for example, `prog1.c` . Then in order to list all files containing C code in your home directory, you need only type `ls *.c` in that directory.

Note that endings indicate the user (and sometimes also some programs) to which kind of group or application the files belongs. In Linux, there is a special command `file` that determines the type of the file, for instance "PDF document", disregard of the ending of the file.

1.6.3 Getting Help

On-line Manuals

There are on-line manuals which gives information about most commands. The manual pages tell you which options a particular command can take, and how each option modifies the behaviour of the command. Type `man` command to read the manual page for a particular command.

For example, to find out more about the `wc` (word count) command, type

```
prompt:  man wc
```

Type `q` to exit. Alternatively

```
prompt:  whatis wc
```

gives a one-line description of the command, but omits any information about options etc.

Apropos

When you are not sure of the exact name of a command,

```
prompt:  apropos keyword
```

will give you the commands with keyword in their manual page header. For example, try typing

```
prompt:  apropos copy
```

1.6.4 Summary

`*`	match any number of characters
`?`	match one character
`man command`	read the online manual page for a command
`whatis command`	brief description of a command
`apropos keyword`	match commands with keyword in their man pages

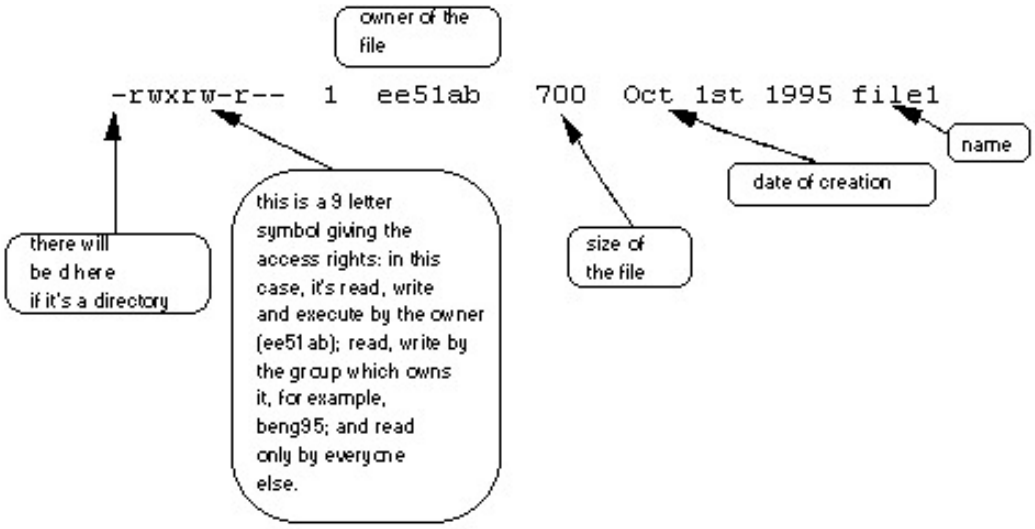
1.7 File Permissions and Process Handeling

1.7.1 File system security (access rights)

In your unixstuff directory, type

```
prompt:  ls -l (l for long listing!)
```

You will see that you now get lots of details about the contents of your directory, similar to the example below.



Each file and directory has associated access rights, which may be found by typing `ls -l`:

```
-rwxrw-r-- 1 ee51ab beng95 2450 Sept29 11:52 file1
```

In the left-hand column is a 10 symbol string consisting of the symbols d, r, w, x, -, and, occasionally, s or S. If d is present, it will be at the left hand end of the string, and indicates a directory: otherwise - will be the starting symbol of the string.

The 9 remaining symbols indicate the permissions, or access rights, and are taken as three groups of 3.

- The left group of 3 gives the file permissions for the user that owns the file (or directory) (ee51ab in the above example);
- the middle group gives the permissions for the group of people to whom the file (or directory) belongs (eebeng95 in the above example);
- the rightmost group gives the permissions for all others.

The symbols r, w, etc., have slightly different meanings depending on whether they refer to a simple file or to a directory.

Access rights on files:

- r indicates read permission, that is, the presence of permission to read and copy the file
- w indicates write permission, that is, the permission to change a file
- x indicates execution permission, that is, the permission to execute a file, where appropriate

Access rights on directories:

- r allows users to list files in the directory;
- w means that users may delete files from the directory or move files into it;
- x means the right to access files in the directory. This implies that you may read files in the directory provided you have read permission on the individual files.

So, in order to read a file, you must have execute permission on the directory containing that file, and hence on any directory containing that directory as a subdirectory, and so on, up the tree.

Some examples

`-rwxrwxrwx`	a file that everyone can read, write and execute (and delete).
`-rw-------`	a file that only the owner can read and write - no-one else can read or write and no-one has execution rights (e.g. your mailbox file).

1.7.2 Changing access rights

Only the owner of a file can use `chmod` to change the permissions of a file. The options of `chmod` are as follows

Symbol	Meaning
u	user
g	group
o	other
a	all
r	read
w	write (and delete)
x	execute (and access directory)
+	add permission
-	take away permission

For example, to remove read write and execute permissions on the file `biglist` for the group and others, type

```
prompt:  chmod go-rwx biglist
```

This will leave the other permissions unaffected.

To give read and write permissions on the file `biglist` to all,

```
prompt:  chmod a+rw biglist
```

Exercise 5a: Try changing access permissions on the file `science.txt` and on the directory `backups`. Use `ls -l` to check that the permissions have changed.

1.7.3 Processes and Jobs

A process is an executing program identified by a unique PID (process identifier). To see information about your processes, with their associated PID and status, type

```
prompt:  ps
```

A process may be in the foreground, in the background, or be suspended. In general the shell does not return the UNIX prompt until the current process has finished executing.

Some processes take a long time to run and hold up the terminal. Backgrounding a long process has the effect that the UNIX prompt is returned immediately, and other tasks can be carried out while the original process continues executing.

Running background processes

To background a process, type an & at the end of the command line. For example, the command sleep waits a given number of seconds before continuing. Type

```
prompt:  sleep 10
```

This will wait 10 seconds before returning the command prompt %. Until the command prompt is returned, you can do nothing except wait.

To run sleep in the background, type

```
prompt:  sleep 10 &

[1] 6259
```

The & runs the job in the background and returns the prompt straight away, allowing you do run other programs while waiting for that one to finish.

The first line in the above example is typed in by the user; the next line, indicating job number and PID, is returned by the machine. The user is be notified of a job number (numbered from 1) enclosed in square brackets, together with a PID and is notified when a background process is finished. Backgrounding is useful for jobs which will take a long time to complete.

1.7.4 Backgrounding a current foreground process

At the prompt, type

```
prompt:  sleep 100
```

You can suspend the process running in the foreground by holding down the `[control]` key and typing `[z]` (written as `^Z`) Then to put it in the background, type

```
prompt:  bg
```

Do not background programs that require user interaction on the terminal.

1.7.5 Listing suspended and background processes

When a process is running, backgrounded or suspended, it will be entered onto a list along with a job number. To examine this list, type

```
prompt:  jobs
```

An example of a job list could be

```
[1] Suspended sleep 100
[2] Running netscape
[3] Running nedit
```

To restart (foreground) a suspended processes, type

```
prompt:  fg %[jobnumber]
```

For example, to restart `sleep 100`, type

```
prompt:  fg %1
```

Typing `fg` with no job number foregrounds the last suspended process.

1.7.6 Running Graphical Processesin the Background

The running jobs in the foreground or background or suspending a job becomes more obvious when you are running graphical jobs. A good example is the text editor `nedit`. When you type

```
prompt:  nedit
```

a window should open in which you can type. You can try to write for instance your name into this window. The terminal window is however blocked, i.e. there is no prompt in the active line where there cursor is and the commands you type do not have any effect. For instance, when you type `ls [ENTER]` nothing happens. If you suspend the job using `[CTRL]+[z]` the terminal is active again, but you can not type anything in the editor window. Try to type something in the window where you wrote your name. You will see, nothing happens. If you

get the job in the foreground again by typing `fg [ENTER]` in your terminal window, the editor is active again, but the terminal is blocked. Try it!

Suspending the job (`[CTRL]+[z]`) and shifting it in the background (`bg [ENTER]`) will make the terminal active and your editor functioning. Try! Close the editor now through the editors menu. If you restart your editor now with

```
prompt:  nedit &
```

The job is immediately started in the background.

1.7.7 Killing a process

kill (terminate or signal a process)

It is sometimes necessary to kill a process (for example, when an executing program is in an infinite loop)

To kill a job running in the foreground, type `[CTRL]+[c]`. For example, run

```
prompt:  sleep 10000
^C
```

To kill a suspended or background process, type

```
prompt:  kill %[jobnumber]
```

For example, run

```
prompt:  sleep 10000 &
prompt:  jobs
```

If it is job number 4, type

```
prompt:  kill %4
```

To check whether this has worked, examine the job list again to see if the process has been removed.

ps (process status)

Alternatively, processes can be killed by finding their process numbers (PIDs) and using kill PID_number

```
prompt:  sleep 1000 &
prompt:  ps


  PID TT S TIME COMMAND
  20077 pts/5 S 0:05 sleep 1000
```

```
21563 pts/5 T 0:00 netscape
21873 pts/5 S 0:25 nedit
```

To kill off the process sleep 100, type

```
prompt:  kill 20077
```

and then type ps again to see if it has been removed from the list.

If a process refuses to be killed, uses the -9 option, i.e. type

```
prompt:  kill -9 20077
```

Note: It is not possible to kill other users' processes !!!

1.7.8 Summary

`ls -lag`	list access rights for all files
`chmod [options] file`	change access rights for named file
`command &`	run command in background
`^C`	kill the job running in the foreground
`^Z`	suspend the job running in the foreground
`bg`	background the suspended job
`jobs`	list current jobs
`fg %1`	foreground job number 1
`kill %1`	kill job number 1
`ps`	list current processes
`kill 26152`	kill process number 26152

1.8 Some More Useful Commands

df

The `df` command reports on the space left on the file system. For example, to find out how much space is left on the fileserver, type

```
prompt:  df .
```

du

The `du` command outputs the number of kilobyes used by each subdirectory. Useful you want to find out which directory consumes most diskspace. In your home-directory, type

```
prompt:  du
```

gzip

This also compresses a file, and is more efficient than an older program `compress` . For example, to zip `science.txt`, type

```
prompt:  gzip science.txt
```

This will zip the file and place it in a file called science.txt.gz
To unzip the file, use the gunzip command.

```
prompt:  gunzip science.txt.gz
```

file

`file` classifies the named files according to the type of data they contain, for example ASCII (simple text), pictures, compressed data, etc.. To report on all files in your home directory, type

```
prompt:  file *
```

history

The C shell keeps an ordered list of all the commands that you have entered. Each command is given a number according to the order it was entered.

`prompt:  history` (show command history list)

If you are using the C shell, you can use the exclamation character (!) to recall commands easily.

`prompt:  !!` (recall last command)
`prompt:  !-3` (recall third most recent command)
`prompt:  !5` (recall 5th command in list)
`prompt:  !grep` (recall last command starting with grep)
You can increase the size of the history buffer by typing

```
prompt:  set history=100
```

1.9 Compiling UNIX software packages

We have many open source software packages installed on our systems, which are available to all users. However, students are allowed to download and install small software packages in their own home directory, software usually only useful to them personally.

There are a number of steps needed to install the software.

- Locate and download the source code (which is usually compressed)
- Unpack the source code
- Compile the code
- Install the resulting executable
- Set paths to the installation directory

Of the above steps, probably the most difficult is the compilation stage.

Compiling Source Code

All high-level language code must be converted into a form the computer understands. For example, C language source code is converted into a lower-level language called assembly language. The assembly language code made by the previous stage is then converted into object code which are fragments of code which the computer understands directly. The final stage in compiling a program involves linking the object code to code libraries which contain certain built-in functions. This final stage produces an executable program.

To do all these steps by hand is complicated and beyond the capability of the ordinary user. A number of utilities and tools have been developed for programmers and end-users to simplify these steps.

make and the Makefile

The `make` command allows programmers to manage large programs or groups of programs. It aids in developing large programs by keeping track of which portions of the entire program have been changed, compiling only those parts of the program which have changed since the last compile.

The `make` program gets its set of compile rules from a text file called `Makefile` which resides in the same directory as the source files. It contains information on how to compile the software, e.g. the optimisation level, whether to include debugging info in the executable. It also contains information on where to install the finished compiled binaries (executables), manual pages, data files, dependent library files, configuration files, etc.

Some packages require you to edit the Makefile by hand to set the final installation directory and any other parameters. However, many packages are now being distributed with the GNU configure utility.

configure

As the number of UNIX variants increased, it became harder to write programs which could run on all variants. Developers frequently did not have access to every system, and the characteristics of some systems changed from version to version. The GNU configure and build system simplifies the building of programs distributed as source code. All programs are built using a simple, standardised, two step process. The program builder need not install any special tools in order to build the program.

The `configure` shell script attempts to guess correct values for various system-dependent variables used during compilation. It uses those values to create a `Makefile` in each directory of the package.

The simplest way to compile a package is:

- `cd` to the directory containing the package's source code.
- Type `./configure` to configure the package for your system.
- Type `make` to compile the package.
- Optionally, type `make check` to run any self-tests that come with the package.
- Type `make install` to install the programs and any data files and documentation.
- Optionally, type `make clean` to remove the program binaries and object files from the source code directory

The configure utility supports a wide variety of options. You can usually use the `--help` option to get a list of interesting options for a particular configure script.

The only generic options you are likely to use are the `--prefix` and `--exec-prefix` options. These options are used to specify the installation directories.

The directory named by the `--prefix` option will hold machine independent files such as documentation, data and configuration files.

The directory named by the `--exec-prefix` option, (which is normally a subdirectory of the `--prefix` directory), will hold machine dependent files such as executables.

1.9.1 Downloading source code

For this example, we will download a piece of free software that converts between different units of measurements.

First create a download directory

```
prompt:  mkdir download
```

Download the software here [3],

and save it to your new `download` directory.

[3] http://www.ee.surrey.ac.uk/Teaching/Unix/units-1.74.tar.gz

1.9.2 Extracting the source code

Go into your download directory and list the contents.

```
prompt:  cd download
prompt:  ls -l
```

As you can see, the filename ends in `tar.gz`. The `tar` command turns several files and directories into one single tar-file. This file is then compressed using the `gzip` command (to create a `tar.gz` file).

First unzip the file using the `gunzip` command. This will create a `.tar` file.

```
prompt:  gunzip units-1.74.tar.gz
```

Then extract the contents of the tar file.

```
prompt:  tar -xvf units-1.74.tar
```

Again, list the contents of the `download` directory, then go to the `units-1.74` sub-directory.

```
prompt:  cd units-1.74
```

1.9.3 Configuring and creating the Makefile

The first thing to do is carefully read the `README` and `INSTALL` text files (use the `less` command). These contain important information on how to compile and run the software.

The units package uses the GNU configure system to compile the source code. We will need to specify the installation directory, since the default will be the main system area which you will not have write permissions for. We need to create an install directory in your home directory.

```
prompt:  mkdir ~/units174
```

Then run the configure utility setting the installation path to this.

```
prompt:  ./configure --prefix=$HOME/units174
```

NOTE:

The $HOME variable is an example of an environment variable. The value of $HOME is the path to your home directory. Just type

```
prompt:  echo $HOME
```

to show the contents of this variable. We will learn more about environment variables in a later chapter.

If configure has run correctly, it will have created a Makefile with all necessary options. You can view the Makefile if you wish (use the `less` command), but do not edit the contents of this.

1.9.4 Building the package

Now you can go ahead and build the package by running the make command.

```
prompt:  make
```

After a few seconds (depending on the speed of the computer), the executables will be created. You can check to see everything compiled successfully by typing

```
prompt:  make check
```

If everything is okay, you can now install the package.

```
prompt:  make install
```
This will install the files into the ~/units174 directory you created earlier.

1.9.5 Running the software

You are now ready to run the software (assuming everything worked).

```
prompt:  cd ~/units174
```

If you list the contents of the units directory, you will see a number of subdirectories.

- bin — The binary executables
- info — GNU info formatted documentation
- man — Man pages
- share — Shared data files

To run the program, change to the bin directory and type

```
prompt:  ./units
```

As an example, convert 6 feet to metres.

```
You have:  6 feet
You want:  metres
* 1.8288
```

If you get the answer 1.8288, congratulations, it worked.

To view what units it can convert between, view the data file in the share directory (the list is quite comprehensive).

To read the full documentation, change into the info directory and type

```
prompt:  info --file=units.info
```

1.10 Variables and General Settings

1.10.1 UNIX Variables

Variables are a way of passing information from the shell to programs when you run them. Programs look ”in the environment” for particular variables and if they are found will use the values stored. Some are set by the system, others by you, yet others by the shell, or any program that loads another program.

Standard UNIX variables are split into two categories, environment variables and shell variables. In broad terms, shell variables apply only to the current instance of the shell and are used to set short-term working conditions; environment variables have a farther reaching significance, and those set at login are valid for the duration of the session. By convention, environment variables have UPPER CASE and shell variables have lower case names.

1.10.2 Environment Variables

An example of an environment variable is the `OSTYPE` variable. The value of this is the current operating system you are using. Type

```
prompt:  echo $OSTYPE
```

More examples of environment variables are

- `USER` (your login name)
- `HOME` (the path name of your home directory)
- `HOST` (the name of the computer you are using)
- `ARCH` (the architecture of the computers processor)
- `DISPLAY` (the name of the computer screen to display X windows)
- `PRINTER` (the default printer to send print jobs)
- `PATH` (the directories the shell should search to find a command)

Finding out the current values of these variables

ENVIRONMENT variables are set using the `setenv` command, displayed using the `printenv` or `env` commands, and unset using the unsetenv command.

To show all values of these variables, type

```
prompt:  printenv | less
```

1.10.3 Shell Variables

An example of a shell variable is the history variable. The value of this is how many shell commands to save, allow the user to scroll back through all the commands they have previously entered. Type

```
prompt:  echo $history
```

More examples of shell variables are

- `cwd` (your current working directory)
- `home` (the path name of your home directory)
- `path` (the directories the shell should search to find a command)
- `prompt` (the text string used to prompt for interactive commands shell your login shell)

Finding out the current values of these variables.

SHELL variables are both set and displayed using the `set` command. They can be unset by using the unset command.

To show all values of these variables, type

```
prompt:  set | less
```

So what is the difference between PATH and path ?

In general, environment and shell variables that have the same name (apart from the case) are distinct and independent, except for possibly having the same initial values. There are, however, exceptions.

Each time the shell variables home, user and term are changed, the corresponding environment variables HOME, USER and TERM receive the same values. However, altering the environment variables has no effect on the corresponding shell variables.

PATH and path specify directories to search for commands and programs. Both variables always represent the same directory list, and altering either automatically causes the other to be changed.

1.10.4 Using and setting variables

Each time you login to a UNIX host, the system looks in your home directory for initialisation files. Information in these files is used to set up your working environment. The C and TC shells uses two files called .login and .cshrc (note that both file names begin with a dot).

At login the C shell first reads `.cshrc` followed by `.login`

`.login` is to set conditions which will apply to the whole session and to perform actions that are relevant only at login.

`.cshrc` is used to set conditions and perform actions specific to the shell and to each invocation of it.

The guidelines are to set ENVIRONMENT variables in the `.login` file and SHELL variables in the `.cshrc` file.

WARNING: NEVER put commands that run graphical displays (e.g. a web browser) in your .cshrc or .login file.

In the practical, your settings are saved in .cshrc.own, while .cshrc is a link to a file in our directories, which helps us to have the same settings for all students.

1.10.5 Setting shell variables in the .cshrc file

For example, to change the number of shell commands saved in the history list, you need to set the shell variable history. It is set to 100 by default, but you can increase this if you wish.

```
prompt:  set history = 200
```

Check this has worked by typing

```
prompt:  echo $history
```

However, this has only set the variable for the lifetime of the current shell. If you open a new `konsole` window, it will only have the default history value set. To PERMANENTLY set the value of history, you will need to add the set command to the .cshrc.own file.

First open the `.cshrc.own` file in a text editor. An easy, user-friendly editor to use is `nedit`.

```
prompt:  nedit ~/.cshrc.own
```

write the following line if the file is empty (or add it to the end of the file in case something is written already in this file).

```
set history = 200
```

Save the file and force the shell to reread its `.cshrc.own` file buy using the shell source command.

```
prompt:  source ~/.cshrc.own
```

Check this has worked by typing

```
prompt:  echo $history
```

1.10.6 Setting the path

When you type a command, your path (or PATH) variable defines in which directories the shell will look to find the command you typed. If the system returns a message saying ”command: Command not found”, this indicates that either the command doesn't exist at all on the system or it is simply not in your path.

For example, to run units, you either need to directly specify the units path (`~/units174/bin/units`), or you need to have the directory `~/units174/bin` in your path.

You can add it to the end of your existing path (the `$path` represents this) by issuing the command:

```
prompt:  set path = ($path ~/units174/bin)
```

Test that this worked by trying to run units in any directory other that where units is actually located.

```
prompt:  cd; units
```

HINT: You can run multiple commands on one line by separating them with a semicolon.

To add this path PERMANENTLY, add the following line to your `.cshrc` AFTER the list of other commands.

```
set path = ($path ~/units174/bin)
```

1.11 Editors

There are several Text-editors (ASCII-editors) under Linux. The most common editors that are used by programmers are `vi` and `emacs`. These editors are however a bit more difficult to use. An easier editor is for instance `kwrite` or `nedit`.

You are free to use whatever text editor (ASCII editor) in the practical (it has to be installed on the system). Note, LibreOffice is NOT an ASCII-editor, but a word processor. Start the text editor of your choice (i.e. for instance `emacs` or `nedit` but NOT `libreoffice`) and try to get a bit familiar with it.

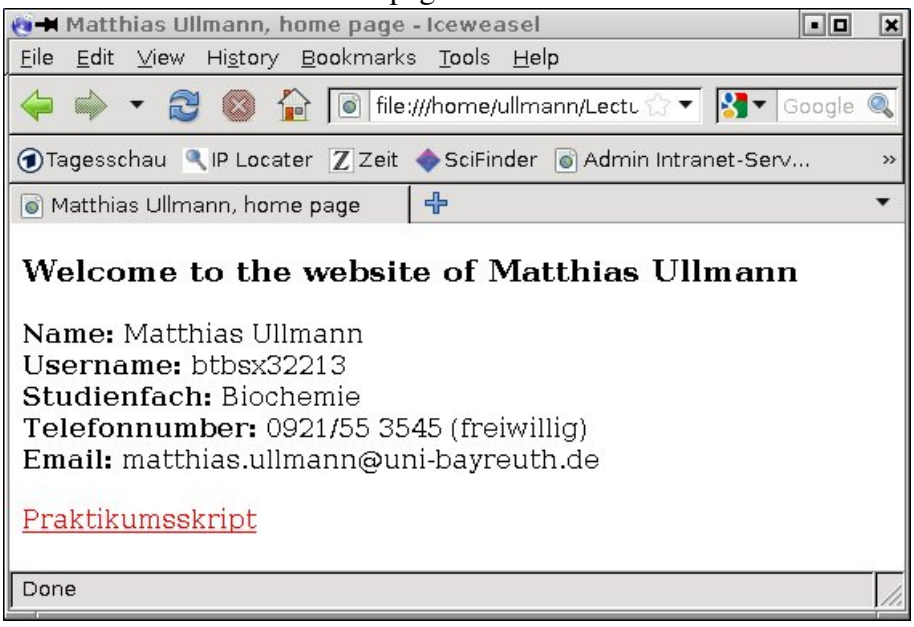
1.12 Webpages

Now your task is to make a small webpage, i.e. a HTML-Page, using an text editor. You find general information on HTML and webpage development at selfhtml.org [4]. In particular here [5], you find some useful information on XHTML and differences to HTML.

Maybe needed/helpful:

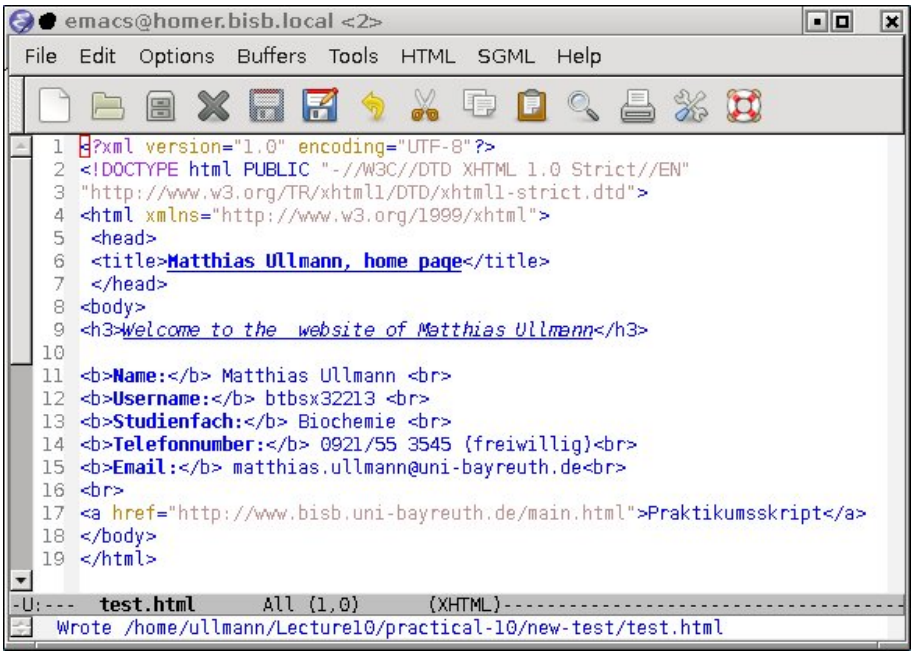
- Encoding of umlauts [6],
- HTML reference [7],

Your task is to write a webpage that looks like this in a webbrowser (firefox or so):



The html-source of this page looks like this (in emacs).

[4] http://de.selfhtml.org/
[5] http://de.selfhtml.org/html/xhtml/unterschiede.htm
[6] http://de.selfhtml.org/html/allgemein/zeichen.htm#umlaute
[7] http://de.selfhtml.org/navigation/html.htm



Type the HTML-code of the page in an ASCII-Editor and replace the personal information with your contact data (You do not have to give your telephone number, but it might be useful if we want to contact you because of the protocol). Pay attention that you also replace the title of the page! Write the correct link to the practical page into the webpage. Save the page as "index.html". Further Reading:

- XML Guide [8],
- XML annotated RFC [9],

Tasks:

- Legen Sie einen Ordner `Praktikum` und Unterordner `Tag_1` bis `Tag_5` an (Exakte Schreibweise!). Alle Ordner müssen für Gruppenmitglieder lesbar und ausführbar sein (Kommando `chmod`). Die Ordner dürfen nicht mehr Rechte haben als erforderlich (z.B. nicht schreibbar für Gruppenmitglieder oder lesbar für alle).
- Speichern Sie die erstellte "Homepage" unter dem Namen `index.html` im Verzeichnis `$HOME/Praktikum/Tag_1/Protokoll/` und bewegen Sie alle Daten (Kommando `mv`), die Sie heute erzeugt bzw. bearbeitet haben, in das Verzeichnis `$HOME/Praktikum/Tag_1/` (Lese-Rechte!)
- Ein eigentliches Protokoll ist für diesen Tag nicht notwendig. Den Punkt gibt es für die richtige Verzeichnisstruktur und Rechtevergabe sowie für das Bewegen der Dateien. Entsprechend kritisch wird kontrolliert. Die grafische Benutzerschnittstelle liefert wahrscheinlich falsche Ergebnisse.

[8] http://www.xml.com/pub/a/98/10/guide0.html
[9] http://www.xml.com/axml/testaxml.htm

Chapter 2

Sequences and Databases

Please use the directory `$HOME/Praktikum/Tag_2` as you working directory for these exercises.

Basic use of Databases and Bioinformatic Analysis using Emboss

In this course, we will learn how to analyze sequences and how to use databases to obtain more information on proteins There are many databases on the web. A long list is published every January in the journal Nuclei Acid Research and accessible on this web site. [1] Also every year (in July), Nuclei Acid Research publishes a list of webservers. This list is accessible on this web site. [2] We will come back to some databases and webservers later in this course.

In this excercise, we will examine a circular DNA sequences. The sequence is given in this file: unknown-cDNA.fasta. [3] Please download this file to your working directory.

Now we will analyze this sequence (download the file!):

```
> unkonwn circular DNA
TCAAATCTTCTTAGATGATGAACAGATGGGTAAAGGCTTCGCTCTACTTTGTGTTACTTACCCTCGTTCC
AACTGCACAATTAAGACCCACCAAGAACCGTACCTTGCTTAATTCATTGCTGTAGTCGCTACTATTTACA
GCTTGTGCAAGTGTAGCTTAAATCAGGATGAAGGATATTTTTCATCCTGATTTCTTATATAAATCTATTC
TGAAACGTTGCAATACAGGACTAGAACTTATCCAAAGAACATCGAGATCATCAGACATAAAGACACCAAA
TTATTAGGAATTAATACTGATCAATAGTCAATAATTCATAATCAACAGTCCATAGTAAGAGCTATAAACT
.
.
.
TAGAGGCTAGCAGCTACAGACTAGTCCCTAGCCTTTCTGTTTTTTCACTCTGTGTCTACATCCAATTAAA
ACCAGTGTCCAACTCTGGCTTAATTCTCGAAGAATTCAGGGAGATTTAACTTCGAGAACTATATCGATCC
TATAAATTTGAGGAGAATCGGCAAATGGCTAGCTACCAAGTTAGATTGATCAACAAGAAACAAGACATCG
ATACTACCATCGAGATTGATGAAGAAACCACAATTTTAGATGGCGCAGAAGAAAATGGTATTGAATTACC
TTTCTCTTGCCATTCTGGTTCTTGTTCTAGCTGTGTAGGCAAAGTTGTTGAAGGTGAAGTTGACCAATCT
GA
```

You can check with the commands `less` or `more` if the file was downloaded correctly.

[1] http://www.oxfordjournals.org/nar/database/c/

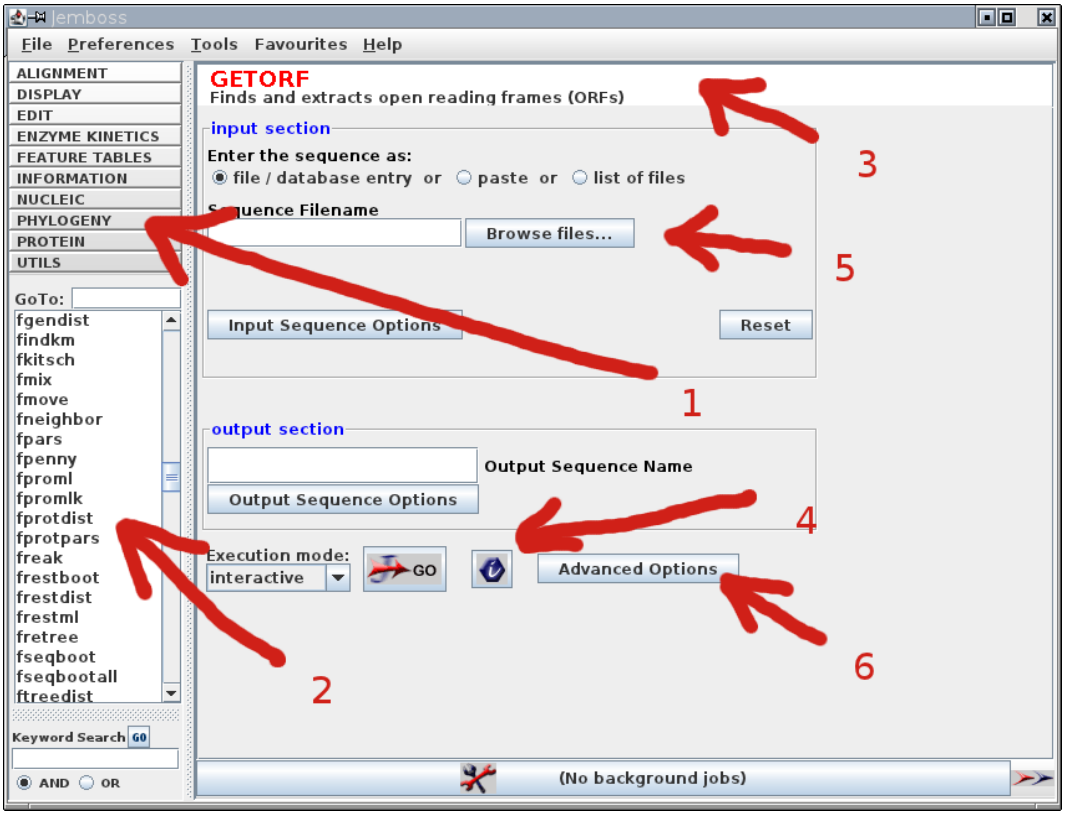
[2] http://bioinformatics.ca/links_directory

[3] http://www.bisb.uni-bayreuth.de/People/ullmann/pract/unknown-cDNA.fasta

2.1 Find ORFs in the Sequence

Now we want to see if there are region in this piece of DNA that code for protein, thus one needs to look for open reading frames (ORF). We will use the software suite EMBOSS for the sequence analysis. EMBOSS is a very useful collection of open source programs for sequence analysis of all kind.

Use Google to find the EMBOSS webpage. Read "About EMBOSS" to get a brief overview of the program. The program **`jemboss`** is a graphical user interface for the EMBOSS software suite. Here you see a screen shot of the program **`jemboss`**. You should get this picture by typing **`jemboss`** in your terminal window.



Arrows indicate important points:

1. list of the programs by topic
2. alphabetical list of the programs
3. name of the current program
4. button for additional information on the current program
5. input options
6. advanced input options

All programs can also be used through a command-line interface. Alternatively you may use a Webserver, which is listed on the EMBOSS Website. You can use for instance the site Wageningen Bioinformatics Webportal, Netherlands.

Use the program **`getorf`** to find protein genes (Open Reading Frames) in the sequence of your circular DNA. Use the advanced input options to analyse your circular DNA!

1. You are only interested in genes of protein with more than 80 aminoacids.
2. Restrict your output to the translation of regions between start and stop codons.

If you have done everything correctly, you should have four protein sequences.

Task 1:
Save the resulting file with the protein sequences as **`protein.fasta`** in you working directory **`Praktikum/Tag_2`** and include the file in your protocol.

The program **`plotorf`** makes a graphical representation of the ORFs in a DNA. This program also shows in which of the six reading frames the gene was found.

Task 2:
Use **`plotorf`** to generate a graphical representation of the ORFs und include this picture in your protocol. Which gene was found in which reading frame?

2.2 Annotation of Protein Sequences

There are several widely used databases that are used to annotate protein sequences. One very useful database is Pfam, [4].

Task 3:
Use Pfam to get information of the four genes. Concentrate only on the significant Pfam-Matches. Include in your protocol what you found out about the four sequences.

2.3 Using Sequences to Find Structures

The Protein Data Bank (PDB) at the Research Collaboratory for Structural Bioinformatics (RCSB) [5] is a collection of all published protein and nucleic adic structures. This database has a very powerful user interface. Several searches can be performed. We will now use this database to see if there are structures available for the proteins which we found as genes in the circular DNA.

Make a sequence search in the PDB; use the advance search. Look at the search option at the page. Search using the four protein sequences (keep the default settings). Look at the search

[4] http://pfam.xfam.org/
[5] http://www.rcsb.org/

results ([Reports:]/[BLAST/FASTA/PSI-BLAST Results]). You can download the files through the web-interface. You can go to the PDB home page and simply search the PDB code that you want.

However, many databases allow a non-interactive download, i.e. without web browsers. This feature is very useful if you know which datafiles you need to download or which directories you need to download. Then you can script your downloads. A linux-program to make such download is **`wget`**. From the PDB, you can for instance download structures, if you know the PDB-code of the structure. For instance, to download the structure with the PDB-code 1n62, you can type in your 'konsole'

```
wget http://www.rcsb.org/pdb/files/1n62.pdb
```

Task 4:
Search for structures of the four proteins that you found in your piece of DNA. Do a Sequence Search. Add to your protocol a table in which you list:

- the ORF number in your DNA (as given in your getorf output)
- the structures with the highest scores. There might be more than one structure per sequence. Do not consider mutants.
- PDB Code, information on the protein
- name of the PDB enetry (title)
- resolution of the structure (quality of the data; the smaller the numerical value of the resolution, the better the structure)

Download these PDB files into a new subdirectory called **`PDB`** in your working directory **`Praktikum/Tag_2`**.

2.4 Getting Information on Literature, Downloading Papers

Many or probably all scientific journals allow to download articles from the web in the form of PDF-files. Often, you or your institution (university) needs a subscription to the journal to be able to download the articles as PDF files.

Literature databases help you to find the articles you need.

Pubmed [6] is a database that covers much of the published articles in biological science. You can search this database relatively easily. More advanced searches can profit from so-called MESH terms, a controlled vocabulary to support a very effective search. However, you can also get a direct link from the PDB to Pubmed. From Pubmed, you can often access directly the articles in the scientific journals. The link to the article is normally found at the right upper corner of the webpage. Sometimes, there is no link in Pubmed. In this case, the best is to go to the journal website and search there.

[6] http://www.ncbi.nlm.nih.gov/pubmed/

Task 5:
Download the PDF-files in which the downloaded structures were published. Save these PDF-files in a new subdirectory called **`Paper`** in your working directory **`Praktikum/Tag_2`**. Name the files like this: **`[PDB-code]-paper.pdf`**. The easiest way is probably to go through the PDB-webpage.

Task 6:
Two of your four proteins are different ferredoxins from the same organism (i.e. they have different sequences). Look through the papers that you downloaded and write down the different functional tasks of these two ferredoxins.

Task 7:
A very good review on these proteins was published by Carrillo and Ceccarelli. Search this review paper in Pubmed and download the PDF-file. Save it as **`Praktikum/Tag_2/Paper/review.pdf`**.

Another important literature database is Web of Science. [7] However, this database is not free and requires (institutional) subscription. With this database, you have not only access to the abstract of articles, but you can also see which articles are cited by the found article, and probably more important, which articles cite your article of interest. With this feature, you have access to literature that is relevant for your work, but was published after the article you are looking at.

Task 8:
Search in this database the review article by Carrillo and Ceccarelli. How often was this article cited? Which article cited this paper for the first time? Write down the citation in the format: Author, A. B.; Author, C. D. (YEAR): Title. Journal Vol., xx-yy

2.5 Kegg und Brenda

Two other very useful databases are KEGG [8] and Brenda. [9] The first tries to organize most of the biochemical knowledge, the second contains information about individual enzymes. Go to the KEGG database and then to the sub-database "PATHWAY".

As you know by now, the enzyme ferredoxin-NADP reductase (FNR) is involved in photosynthesis. It transfers electrons from ferredoxin to NADP. Go to photosynthesis page of KEGG to learn more about this reaction.

Task 9:
From where does ferredoxin receive its electron?

[7] http://apps.webofknowledge.com/
[8] http://www.genome.ad.jp/kegg/
[9] http://www.brenda-enzymes.info/

Task 10:
What is the enzyme code of FNR and what is the meaning of the code. You can find the information on the IUBMB Enzyme Nomenclature webpage. To find this page, click on the enzyme and go to the bottom of the page.

Task 11:
For which reaction is the reduced NADPH used in photosynthesis (You may also use other web resources). Which enzymes catalyze this reaction and what is their Enzyme Code. What is the meaning of the code?

Go to the webpage of Brenda [10] which is also linked at the bottom of the previous KEGG-webpage. Brenda gives a lot of information on enzymes and lists much of the current knowledge and links to many databases.

Task 12:
Which synthetic inhibitors exist for FNR? List two.

Task 13:
What is the K_M value of FNR using dibromothymoquinone as substrate? What is the structure of dibromothymoquinone?

2.6 Search with Chemical Structures

Suppose you want to repeat the measurement of the Michaelis-Menten kinetics using dibromothymoquinone as substrate. For this you need to order the molecule from a company, for instance from Sigma-Aldrich. [11] On their webpage, you can also search for structures. Do a **Structure Search** using the JME editor. (Go to the very bottom of the page. There is a link on the left side). Draw the structure and search for the molecule.

Task 14:
What is the price of dibromothymoquinone from this company? Which amount do you get?

Task 15:
What is the expected product of the reduction of dibromothymoquinone? Draw the structure using `chemtool` (or other chemical drawing software like `bkchem`) and include a picture in your protocol.

Similar procedures can be used to make literature searches for instance using SciFinder or structure searches in the Cambridge Structural Database. However, our unviversity has only a limited number of licenses for these databanks which make it difficult to use these databanks in practical courses.

[10] http://www.brenda-enzymes.info/
[11] http://www.sigmaaldrich.com/germany.html

Chapter 3

Sequence Alignments

You will perform pairwise and multiple sequence alignments. You will use programs installed on your computer as well as web services. We will continue to work on the sequences that we were starting to use last week. First we give you some background information to remind you then we start with the alignments.

3.1 Ferredoxin-NADP(H)-reductase

Ferredoxins (Fd) and ferredoxin-NADP(H)-reductases (FNR) are electron transfer partners, which means that they interact with each other to transfer electrons. They are a part of the photosynthetic electron transfer chain. This system is well-studied for the cyanobacterium *Anabaena* (= *Nostoc*) but not for the model plant *Arabidopsis thaliana.*

The figure represents the complex between the FNR (red) and Fd (green) from *Anabaena*. The important residues responsible for their interaction are listed in the table (Mayoral, et al. (2005) Proteins, **59**, 592-602):

FNR	Fd
R16	D67
K75	E94
K138	D68
K290	D23
K293	D59
L76/L78	F65

On the other hand, there is no information available for this complex from *A. thaliana*.

This electron transfer complex is very intesting in many respect. For instance in heterocysts, i.e. cells that are not photosynthetically active, but perform nitrogen fixation, the normal ferredoxin is replaced by a so-called heterocyst ferredoxin. Also on this complex there is not much

information. Moreover, under iron deficiency, *Anabaena* expresses a flavodoxin which contains an FMN as redox cofactor instead of an iron sulfur cluster.

Your task is to derive some information about these complexes using sequence alignments.

3.2 Dot-Plots and Pairwise Sequence Alignments

Last time, you downloaded some PDB files (Code: 1QUF, 1QT9, 1FRD, 1FLV). Use the PDB files now. In addition download the PDB file with the code 1E9M (ferredoxin VI from *Rhodobacter capsulatus*). We wrote a little perl script that converts the sequence in a PDB-file into a FASTA-file. Save the perl-script createfasta.pl [1] and the file aminoacids.txt. [2] to your working directory. Make createfasta.pl executable (chmod +x) Use this script to make FASTA-files from the PDB-files that you have.

```
%createfasta.pl [pdb file] aminoacids.txt > [fasta file]
```

Replace the whole expression in square brackets (including the brackets) by the appropriate file names.

Task 1: Use the EMBOSS-program **dottup** to compare the sequences of ferredoxin, heterocyst ferredoxin, ferredoxin VI from *R. caps.* and flavodoxin. Make all pairwise comparisons. Use the wordsizes 2 and 10. Compare your results. Why are the pictures different for different wordsizes. Include in your protocol the generated dotplots. What can you conclude from these dot plots and why.
Task 2: Use the EMBOSS-program **dotpath** to compare the normal ferredoxin, ferredoxin VI from *R. caps.* and heterocyst ferredoxin. Use the wordsizes 4 and 2. Include in your protocol the generated dotplots. Explain the difference between **dotpath** and **dottup**. Use for instance the information system of jemboss to get this information.
Task 3: Use the EMBOSS-program **needle** to make a pairwise alignment of the normal ferredoxin, heterocyst ferredoxin and ferredoxin VI from *R. caps.*. Use the different substitution matrices EPAM250 and EBLOSUM62. Include the alignments in your protocol. Pay attention that you use the correct fonts to display the alignment ('courier' combined 'preformated text' should be for instance ok) so that the alignment can be seen. Compare the results. Are the alignments with different substitution matrices the same? Discuss your observations.

3.3 Multiple Sequence Alignment for Ferredoxin and FNR

Please read until the end before you proceed!

Task 4: Perform a multiple sequence alignment of ferredoxin as described below.

For that purpose, we need to obtain many sequences, which we will get using BLAST and then we perform a progressive sequence alignment using ClustalOmega.

[1] http://www.bisb.uni-bayreuth.de/People/ullmann/pract/createfasta.pl
[2] http://www.bisb.uni-bayreuth.de/People/ullmann/pract/aminoacids.txt

- Take the ferredoxin sequence from *Anabaena* (PDB code 1QT9). Go to the BLAST page on the NCBI webpage and "blast" using this sequence (use Protein BLAST). Note that now *Anabaena* may appear also under the synonym Nostoc. sp. You can have a look at the taxonomy server to verify this synonym.

- Select sequences from your BLAST search for the multiple sequence alignment. You should use ten sequences in addition to the one from *Anabaena*. Include the sequences of ferredoxin from *Arabidopsis thaliana*. Choose five sequences of other organisms that have a sequence identity of less than 90% and more than 65% and five organisms that have a sequence identity of less than 65%. Make also sure that the sequences you choose do not belong to organisms of the same genera (Gattung). To ensure that, it might be helpful to have a look on the taxonomy report of the results.

- REMARK: In order to find the homologous ferredoxin sequences form *Arabidopsis thaliana*, you need to restrict the search to this organism only.

- Download the sequences in FASTA format (10 sequences plus homologous sequences from *Arabidopsis thaliana*. Include the query sequence.

- Next you should align the sequences using the web interface to the program ClustalOmega provided at the EBI-webpage. This program performs multiple sequence alignments. Use this program for multiple sequence alignment of your ferredoxin sequences. Save the sequence alignment on your hard disk in the clustal-format (=".aln"-format). Give sensible names to the files.

- Visualization of the alignments: You can view the alignment with JALVIEW directly on the webpage (in Results Summary). In order to view also the sequence alignment that you saved on your disk, you can use jemboss (Menu: Tools).

Task 5:
Apply the procedure described above for ferredoxin now to the ferredoxin-NADP(H)-reductase (FNR) from Anabaena (PDB code 1QUF). Search from FNR genes in the genome of *A. thaliana*. Save all the FASTA files as well as alignment files. Find the residues in Fd and FNR from *A. thaliana* (from the sequences with the highest E-value) which are aligned to those from the given table for the *Anabaena* system. Are these residues conserved? Which conclusions can you get?
For the Protocol
Copy the .clustalw-files of the two alignments in your "Protokoll"-directory of today. Include in your protocol a table in which you list the residues of the FNR and Ferredoxin from *A. thaliana* that align with the interacting residues of FNR and Ferredoxin from *Anabaena*.

Chapter 4

Protein Structure Visualization

VMD Tutorial

This tutorial gives you a guide through the main functions of VMD, a molecular graphics program designed for the interactive visualization and analysis of biopolymers such as proteins, nucleic acids, lipids, and membranes. Detailed documentation can be found at this web site web site. [1] In this tutorial, the functions of VMD will be explained briefly in the first chapter. Afterwards you will examine the enzyme ferredoxin:$NADP^+$ reductase (FNR) and binary complexes of this enzyme with ferredoxin and NADP. You will model a ternary complex of FNR with ferredoxin and NADP. After going through all the four chapters, you should be able to make a nice and informative figure of FNR.

For this figure and the answers to the questions in the text, you will get the point for the practical.

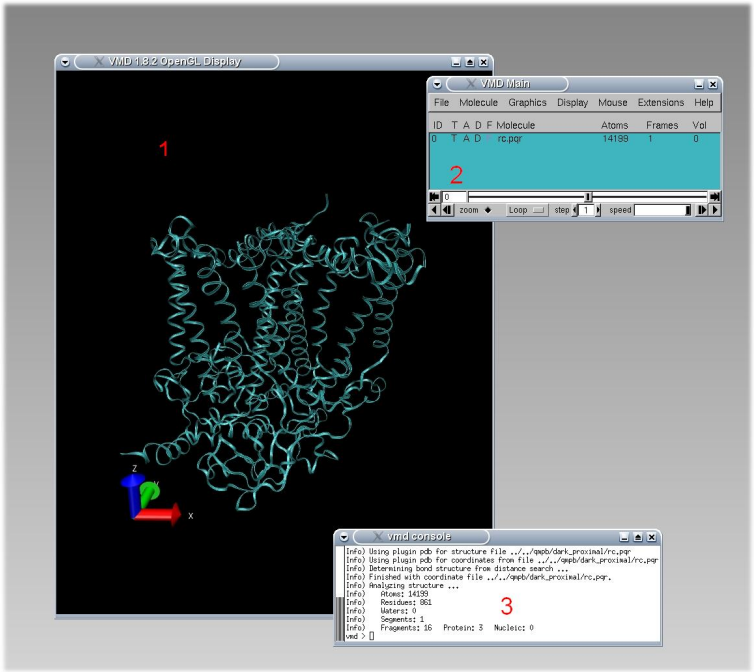
4.1 How to use vmd

To start vmd, type **vmd** in your shell.

1. The three VMD Windows

When you open vmd, three windows will pop up (shown in the figure below):

[1] http://www.ks.uiuc.edu/Research/vmd/current/docs.html



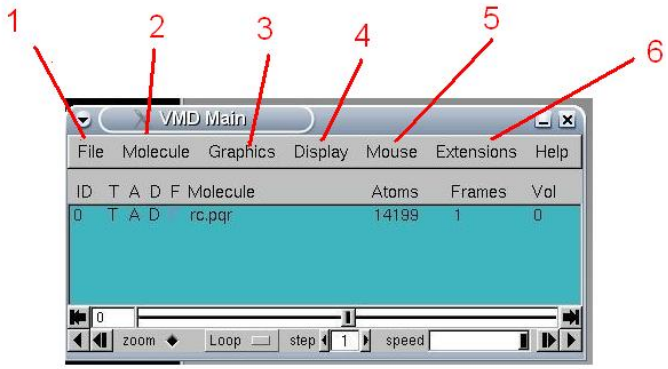
1. **Graphics Display:** This window displays the loaded molecules. By clicking in it you can rotate or translate the protein.

2. **vmd main:** The main window shows you the molecules you have loaded and provides the graphical user interface to all functions of vmd (for more detailed information see below).

3. **vmd console:** All functions available from other windows can also be called from the console by typing appropriate commands. The command ”rotate y by 90” for example, will rotate your molecule by 90 degrees around the y axis. Additionally, it shows information about the program status, the molecule and the function in use. In many Unix systems, the console is not an extra window, but the console from which you started vmd.

2. The vmd main window and its functions

The main menus of the vmd main window are marked in the figure below and are explained in the text.



1. **File:** Here you can load a new molecule (new molecule...) VMD can display multiple molecules at the same time. You can save the currently displayed state (save state...). You can also reopen a previously saved state, restoring all your previously made settings including your loaded molecule, its orientation, its representation, its color and everything else (load state...). You can make a picture by using Render and you can exit the program with Quit.

2. **Molecule:** Besides other functions, you can delete a molecule (delete molecule...).

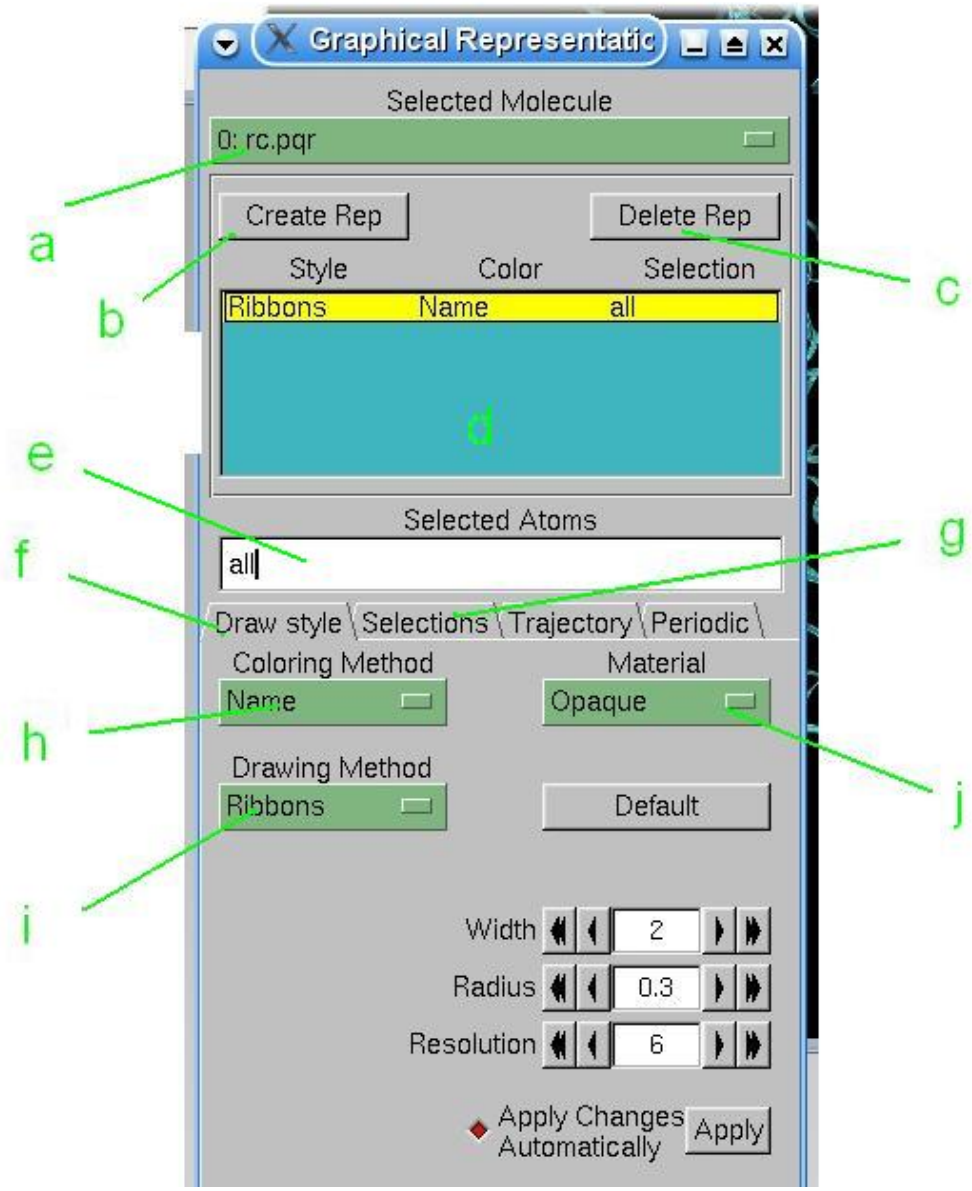
3. **Graphics:** Besides other functions, you can change here the representation (Representation...) of your molecule and the coloring scheme (Colors...) for example for the background or certain atom types and so on. The representation functions will be explained in more detail below.

4. **Display:** Here you can change the view of the protein between perspective and orthographic. Additionally, you can change the light in the OpenGL display by switching on or off light1 to light4.

5. **Mouse:** You can change the function of the mouse in the OpenGL window between translate, rotate, scale and query mode (keyboard shortcuts t, r, s, 0) . In addition, you can label atoms (by pressing 1) and measure distances (by pressing 2), angles (by pressing 3) or dihedral angles (by pressing 4).

6. **Extensions:** Here some helpful tools are located like the calculation of a ramachandran plot or the rsmd between two structures. You can also superimpose two molecules here.

3. The Graphical Representation Window



- **a) Selected Molecule:** Here you can choose for which of the loaded molecules you want to make changes to its representation.
- **b) Create Rep:** with this button you can create a new representation.
- **c) Delete Rep:** lets you delete a representation
- **d) Representation Window:** This window shows you a list of all the representations for the molecule (you might have more than one, for example to see the protein and its cofactors in different representations).
- **e) Selected Atoms:** In this line you specify a selection of atoms for which you want to display a certain representation. The selection window (see g) might help you to type in the right command for selection.

- **f) Draw style:** Here you can change the drawing style of the active representation. In the above figure, this tab is active.
- **g) Selection Shows:** you different ways to select parts of your molecules. Shows for example all residue names found in the molecule. (The functions of this tab are not shown in the figure above).
- **h) Coloring Method:** You can color the selected atoms in different ways. When you click here this, a menu will pop up, where you can choose how to color you representation.
- **i) Drawing Method:** Here you can select how to draw the selected atoms. For example you could draw a cofactor in cpk and the protein in Ribbons. With VDW the selection is shown as interlocking spheres with Van-der-Waals radii.
- **j) Material:** You can choose between opaque and transparent drawing of your representation.

4. A General Hint

The defaults view in vmd is the perspective view, in which objects which are far away are smaller than those nearby. Some people (for instance the author if this text ;-)) might find that disturbing. In the orthographic view, all objects appear at the same scale. You can change this view in the `VMD Main` window, in `Display`.

4.2 The complex of ferredoxin:NADP+ reductase with ferredoxin

Please download the structure of the complex of ferredoxin:$NADP^+$ reductase with ferredoxin from the PDB. The PDB code is 1EWY.

1. Overall structural organization

Start vmd and load in the structure 1ewy.pdb. Change the view of the protein. Open the **`Graphical Representation`**-Window (**Menu:** `Graphics – Representations`). Now color each protein chain differently, use in the `Graphical Representation`-Window `Drawing Method -- Cartoon` and `Coloring Method ->Chain`. How many protein chains can you spot? In this structure, two chain of FNR are cocrystallized with ferredoxin. Can you identify the ferredoxin?

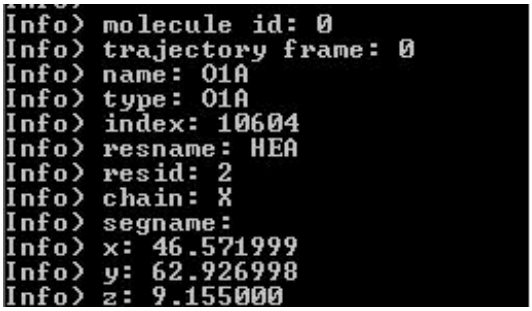
Now, select only ferredoxin! In order to do so, you can go to Selection. There under keywords you will find `chain` and when you click on `chain`, the names of the chains are listed in the window value. Select the chain of ferredoxin.

To make the structure better visable, you may want to choose a different color. To do so, you can choose the color by ColorID.

2. Ferredoxin

Ferredoxin binds a redox-active metal center. In order to identify this center, create a **new representation** and type **not protein and chain X** in `Selected Atoms` (set X to the chainid of ferredoxin) . Change the drawing method to `CPK` and the coloring method to `Name`. Zoom in

(turn the mouse wheel) and identify the residues that are shown as spheres. When you change in the main window `Mouse --- Query` and you click on an atom of the cofactor you will see in the vmd console something like that:



with the following meaning

- **name:** refers th the name of the atom you clicked at.
- **type:** is the atom type of the atom.
- **index:** is the number in the pdb file for your atom.
- **resname:** is the name of the type of residue (like HEA for heme a molecule) you clicked at.
- **resid:** is the number of this residue (every residue has its own number, specifying a certain heme in this case).
- in the end the coordinates of your atom is shown. **(x,y,z)**

The redox-active center is coordinated by four protein residues. Find these residues by again creating a replica of the representation and selecting the residues close to the redox active center. For instance, by typing in `Selected Atoms`: "within 20.0 of (resid 567 and chain R)" you would select all residues within 20.0 Å around the residue with the resid 567 of chain R. You need to adapt this selection to your case.

Task 1: Which redox-cofactor is bound to ferredoxin? List protein residues that coordinate the redox cofactor.

3. FNR

There are two chain of FNR in this crystal structure. Only one of them is most likely in a position that represents the functional situation, i.e., in a orientation in which FNR can accept electrons from ferredoxin. In order to identify the chain that is in the active position, you first need to select the redox-active groups of FNR and then measure the distances between the redox cofactors of the proteins. The shorter the distance, the better the electron transfer activity.

To measure the distance between the two redox-active groups, click in the vmd main window on `Mouse -- Label -- Bonds` and then on atoms of the redox-active groups.

Task 2: What is the redox-active cofactor of FNR? Briefly describe the redox chemistry of this cofactor (reaction scheme and a few sentences). What is the shortest distance (approximate!) between the redox-active sites in FNR and ferredoxin (give the approximate distances between the redox-active site of ferredoxin and the redox-active sites of the two FNRs, use a meaningful number of digits and give a unit!)

Task 3: What is the chainid of the FNR that is most likely in a good position to accept electrons from ferredoxin?

To remove the labels go to Graphics->Labels where you can delete the atom and bond labels! You can select all labels at once by pressing the SHIFT key while selecting the first and last label.

At this point, you know about some basic usage of vmd. You can learn more about it on the VMD webpage. [2]

In particular, it might be good to glance through (not everything needs to be read in detail!):

- Using the Mouse in the Graphics Window. [3]
- Description of each VMD form. [4]
- Molecular Drawing Methods. [5]

4.3 Analysis of the complex of FNR with NADP

The PDB contains structures of *Anabeana* ferredoxin with NADP (PDB code: 1gjr and 2bsa). NADP does not have a favorable conformation for hydride transfer in one of the structures. You can compare the structures by superimposing (= structurally aligning) them.

To superimpose the structure, you can use VMD. Load the two structures into vmd. Select Extensions/Analysing/RMSD Calculator. Align both structures and calculate the RMSD afterwards.

Task 4: In which structure is the hydride transfer easily possible? Why?

Task 5: Which structural change is required in the "inactive" structure to make an easier transfer?

Task 6: Which protein residues of the active site could be functionally important for the catalytic reaction of the enzyme?

[2] http://www.ks.uiuc.edu/Research/vmd/current/docs.html
[3] http://www.ks.uiuc.edu/Research/vmd/current/ug/node30.html
[4] http://www.ks.uiuc.edu/Research/vmd/current/ug/node37.html
[5] http://www.ks.uiuc.edu/Research/vmd/current/ug/node53.html

4.4 Model of the Ternary Complex of FNR, Ferredoxin and NADP

You have realized that there are several structures of FNR complexes, representing several functional and non-functional intermediates. However, no crystal structure of a ternary complex is available. But the information contained in several structures could be combined.

For instance, this shell-script models a structure which combines the information contained in two PDB-files.

(Just for information: A shell script is a plain ASCII text file. It is a list of commands which you would normally have issued yourself on the command line.)

```
#! /bin/csh
#
# C-Shell script for generating a model of an FNR/Fd/NADP-complex
#
#=============================================================
#

# define the prefix of the PDB files are variables
#
set ref  = 1ewy
set nadp = 1gjr

# convert the sequence read from the ATOM-records of the
# PDB-file to a fasta file
#
# the command "head -8" writes the first eight line of the output
#
pdb2fasta $ref.pdb ATOM  | head -8 > $ref.fasta
pdb2fasta $nadp.pdb ATOM > $nadp.fasta

# make a sequence alignment using needle of the EMBOSS package.
# This is the command line usage of the program that you used
# through jemboss
#
# the resulting alignment will be written to the file `align.fasta`
#
needle -asequence $nadp.fasta \
       -bsequence $ref.fasta \
       -gapopen 11 -gapextend 1 \
       -aformat3 fasta \
       -outfile align.fasta

# superimpose (= structurally align) the two structures for which the
# sequence alignment is given in the file `align.fasta`.
#
# The first structure in the `align.fasta` will serve as reference,
# the second structure is superimposed to it.
# The resulting superimposed structure is written to a file called
# 'kabsch.pdb'
#
kabsch align.fasta B  > kabsch-bb.out
```

```
#
# This generates now a model containing the Ferredoxin, NADP (unproductive
# conformation), and FNR from the PDB file 1gjr in an orinetation that is
# superimposed with the FNR in an orientation superimposed with
# the FNR/Fd complex (PDB 1ewy)
#
# the command "grep -E ^".{21}C" selects all line that contain the
# character 'C' in the 21st column of the file (chain-id of the protein)
#
cat kabsch.pdb > model.pdb
cat $ref.pdb | grep -E ^".{21}C" >> model.pdb
```

Task 7: Make a model of the ternary Complex of FNR, Ferredoxin and NADP. The generated model of the ternary should contain the FNR mutant (Y303S) with NADP in the productive conformation and ferredoxin.

You can copy the lines from the script above or you can download `script.csh` [6] and run it directly. In case you copy the lines using the mouse, pay attention the character "^" is not reproduced correctly. Use the keyboard!

4.5 Preparation of a Figure of the Ternary Complex

Task 8: Make a figure of the complex modeled in Task 7 having the following features:

- all cofactors and important residues are highlighted (for instance licorice)
- the protein backbone is shown as a cartoon
- the peptide chains should have different colors
- the overall oriention should be clear, so that the important features can be seen

Include this figure in your protocol and describe what is seen (which color is what etc.). To save ink, you may want to use a white background (Menu: `Graphics -- Colors -- Display -- Background`)

To get the figure:

1. Load the modeled structure into VMD
2. Make all the required operation so that the picture fulfills the requirements above and you like the picture on the screen
3. Click in the main VMD window to `File --- Render...`. A new window will pop up. Change the name and press save. The saved snapshot will be opened in a new window.

[6] http://www.bisb.uni-bayreuth.de/People/ullmann/pract/script.csh

Chapter 5

Continuum Electrostatics

Continuum Electrostatics Calculations using APBS

The electrostatics of a biomolecule can be modeled using the Poisson-Boltzmann equation. This partial differential equation describes the protein as a low-dielectric region with fixed charges (defined by the position of the atoms) in water, a high-dielectric region, with solvated ions. The ions adopt a Boltzmann distribution in the electrostatic field of the protein.

$$\nabla \left[\varepsilon(\mathbf{r})\nabla\phi(\mathbf{r})\right] = -4\pi \left(\rho_p(\mathbf{r}) + \sum_{i=1}^{K} c_i^{\text{bulk}} Z_i e_\text{o} \exp\left(\frac{-Z_i e_\text{o}\phi(\mathbf{r})}{RT}\right)\right)$$

The symbols in the equation have the following meaning:

- ∇ represents the differential operator: $\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})$
- $\phi(\mathbf{r})$ electrostatic potential
- $\varepsilon(\mathbf{r})$: relative permittivity (position dependent "dielectric constant")
- $\rho_p(\mathbf{r})$: charge density due to the protein
- c_i^{bulk} ion concentration in the bulk solution
- Z_i charge number
- e_o elementary charge
- R gas constant
- T temperature
- K number of ionic species

Solving this equation means determining the electrostatic potential $\phi(\mathbf{r})$. Analytically, this equation can only be solved for special cases. For irregular objects such as proteins, the equation needs to be solved numerically, for instance with the help of the finite difference method.

The program `apbs` (Adaptive Poisson-Boltzmann Solver) determines this numerical solution. This numerical solution can be visualized to give an optical impression of the electrostatic potential of a protein. However, it can also be used to calculate electrostatic interaction energies and solvation energies. This practical gives a short introduction into the usage of `apbs`.

5.1 Structure Preparation

In order to perform electrostatic calculations on the biomolecular structure you are interested in, one needs to provide:

- atomic coordinates
- atomic partial charge
- atomic radii

The program `apbs` reads this information in form of a PQR-file, which is a modified PDB file. Atomic partial charges are used to define the biomolecular charge distribution for the Poisson-Boltzmann (PB) equation while the radii are used to construct the dielectric and ionic boundaries.

The PQR-format provides a very simple way to include parameter information by replacing the occupancy and temperature columns of a PDB-format structure file (columns after the xyz-coordinates) with charge (”Q”) and radius (”R”) information.

Here are the first few lines of a PQR-file:

```
ATOM      1  N   ASP A   9      26.776   2.259 -10.932 -0.3200 2.0000
ATOM      2  CA  ASP A   9      28.247   2.418 -11.124  0.3300 2.0000
ATOM      3  C   ASP A   9      28.655   3.886 -11.092  0.5500 1.7000
ATOM      4  O   ASP A   9      29.550   4.269 -10.341 -0.5500 1.4000
...
```

It provides the following information in its columns:

1. line type, important are those starting with ATOM (or HETATM)
2. atom number
3. atom name
4. residue name
5. chain name
6. residue number
7. x-coordinate
8. y-coordinate
9. z-coordinate
10. charge (”Q”)
11. radius (”R”)

The `pdb2pqr` web service and software converts PDB files into PQR format. But some problems can occur:

- Limited ligand support. Atom types which are not included in standard force fields are currently not supported by `pdb2pqr`.
- Inability to "repair" large regions of missing coordinates. Although `pdb2pqr` can fix some missing heavy atoms in sidechains, it does not currently have the (non-trivial) capability to model in large regions of missing backbone or sidechain coordinates.

`pdb2pqr` will also perform hydrogen bond optimization, sidechain rotamer search, limited titration state assignment, and `apbs` input file preparation.

The program can be used at the command line (use a terminal window, called "konsole"):

```
pdb2pqr.py --ff=parse --apbs-input --chain model.pdb model.pqr
```

See the documentation for details [1] (local copy). [2].

Task 1: Copy the PDB file of the complex of FNR with ferredoxin and NADP in the productive position (`model.pdb`) to your working directory. You generated this file in the last practical. Ensure, that you used PDB Code 1ewy and 2bsa for building the model. Otherwise, correct it before continuing. Run `pdb2pqr`. What is the meaning of each option of the program? What is the difference between the input PDB-File and the output PQR-File (e.g. using `tkdiff` and/or `vmd`)? Give at least 4 fundamental differences (not single added/removed atom).

The program `tkdiff` highlights differences between similar files. It can be used with the following command line:

```
tkdiff model.pdb model.pqr
```

You also may look at the files with `vmd` (see last practical!). **Remark: Open the PQR-files as PDB because** `vmd` **fails to read the PQR format of** `pdb2pqr`**!** (`File -- New molecule...` in the `VMD Main` window, choose `Determine file type -- PDB`). Alternatively, you can read the file when you start `vmd`:

```
vmd -pdb model.pqr
```

If you have a look at the PQR-file, you realize that it does not contain the iron sulfur center. `pdb2pqr` has problems to handle such metal clusters, because the metals are coordinated by protein residues which therefore also have changed charges. Such changes will now be done by hand.

We calculated these charges for the oxidized form (Fe(III)-Fe(III)) for you using density functional theory:

```
Total Charge: -2
Atom  Residue Charge Radius
 FE1  FES     0.600  1.25
```

[1] http://www.poissonboltzmann.org/pdb2pqr/user-guide/using-pdb2pqr

[2] http://www.bisb.uni-bayreuth.de/People/essigke/pract/pdb2pqr_userguide.html

```
 FE2  FES     0.600  1.25
  S1  FES    -0.600  1.85
  S2  FES    -0.600  1.85

  CB  CYS     0.050  2.00
  SG  CYS    -0.550  1.85
  HB3 CYS     0.000  0.00
  HB2 CYS     0.000  0.00
```

Warning: It is expected that you understand what you are doing! The description will be less and less detailed and you have to transfer your knowlege from the steps you did before! Do not continue, if you get an error message. Try to understand what the error message means and try to solve the problem. Check every step! If you create a file, look if it exists (`ls -rtl`, for the options `man ls`) and check the content (`less filename`, see `man less` how to use it). You may revisit the Unix tutorial of day 1, e.g. how redirections and pipes work.

Edit a copy of the PQR-file to incorporate the iron sulfur cluster. To do this:

- Make a copy of your PQR-file

  ```
  cp model.pqr model_task2.pqr
  ```

- Copy the coordinates of the iron sulfur cluster from the `model.pdb` into your new file. Therefore, get the coordinates from the modeled complex, for instance using the Unix command:

  ```
  grep FES model.pdb | grep -v LINK >> model_task2.pqr
  ```

Task 2: Explain what the line above does. If you do not know, search the `man`-page and the Unix tutorial for answers!

- Try to recall the coordination of the iron-sulfur cluster which you studied in the last practical. Work only on the cysteines which coordinate the iron-sulfur cluster!
- Open `model_task2.pqr` in an ASCII text editor, e.g. `kwrite` or `nedit` (NOT `openoffice`!).
- Delete the thiol hydrogens of the four coordinating cysteines.
- Add the charges and radii of the iron sulfur cluster in the PQR-file. Correct the charges of the cysteine atoms. Take the values from the table above. Check with

  ```
  grep FES model_task2.pqr
  ```

  that all atoms of the iron-sulfur cluster have proper charges and radii. Check with

```
grep "CYS C" model_task2.pqr | grep -E "CB|SG|HB2|HB3"
```

that all coordinating cysteines have proper charges and radii. `grep -E` allows to use regular expressions. The symbol `"|"` in `"CB|SG|HB2|HB3"` is a logical `"or"`. It selects the line if one of the four alternative atom names is given in the current line.

- Delete all water molecules from your input by using the Unix command

```
grep -v HOH model_task2.pqr | grep -v REMARK > model-mod.pqr
```

- Compare `model-mod.pqr` with `model.pqr` using `tkdiff`. Make sure you find all changes you intended to make, and no unintended changes.
- Check the total charge to be a realistic integer (-1000 is not realistic!). The following command adds up the charge column of your PQR-file:

```
grep -E ^"ATOM|HETATM" model-mod.pqr | \
        awk ' BEGIN {charge = 0} {charge+= $10} END {print charge}'
```

The program `grep` is used to select lines containing `ATOM` or `HETATM` at the beginning of a line (`'^ '`). These lines are piped into a script in the language `awk` to sum up the 10th column ($10). Before the first line is processed, the variable `charge` is set to zero, the variable is incremented for each line by the value of the 10th column and after the last line the value of the variable is printed as output. Awk may do rounding mistakes. Therefore the last digit of the result can be ignored. If your result is not a reasonable integer, check your changes again!

- You may have realized that FAD and NADP are missing in the PQR file, since `pdb2pqr` can not not deal with them. You can download a PQR file here. [3]. Add these molecules to you PQR file. You can do that for instance by using the program `"cat"`. Call the resulting file `complex.pqr`.
- Check again that everything is correct!

Task 3: What is the total charge of FAD, NADP and the complex? In case of FAD and NADP: on which groups are the charges formally located?

In the following, we will compare the electrostatic potentials of the complex and of its components. For this purpose, we need to generate several PQR-Files for:

- FNR alone with bound FAD. Get FNR with FAD from your generated PQR-file, for instance by typing

```
grep -E ^".{21}A"  complex.pqr  >  fnr.pqr
grep -E FAD complex.pqr         >> fnr.pqr
```

[3] http://www.bisb.uni-bayreuth.de/People/essigke/pract/fad_nap.pqr

- Ferredoxin, get the PQR-file by using

```
grep -E  ^".{21}C"  complex.pqr > fd.pqr
```

- FNR alone with bound FAD and NADP using similar commands (target file `fnr-nap.pqr`).

Do the molecules have the charge which you would expect?

It is very important that you are using a consistent naming scheme of your files. Otherwise you will be confused very fast. For example use the following filenames:

Task	PQR-File	`apbs` input	`apbs` output	DX-File	isosurface	surface potential
4	fd.pqr	fd.in	fd.out	fd.dx		
5					fd-iso.tga	
6						fd-surf.tga
7	fd.pqr	fd-salt.in	fd-salt.out	fd-salt.dx	fd-iso-salt.tga	fd-surf-salt.tga
8	fnr.pqr	fnr-salt.in	fnr-salt.out	fnr-salt.dx		fnr-surf-salt.tga
	fnr-nap.pqr	fnr-nap-salt.in	fnr-nap-salt.out	fnr-nap-salt.dx		fnr-nap-surf-salt.tga
	complex.pqr	complex-salt.in	complex-salt.out	complex-salt.dx		complex-surf-salt.tga

5.2 Running the Electrostatics Calculation

You are now almost ready for electrostatics calculations.

Have a look at the `apbs` input file that you have created with `pdb2pqr`. Try to understand what is written there using the `apbs` documentation.

The input file is structured as follows

- First `apbs` reads in the PQR-file.
- Then `apbs` does the electrostatic calculation. The `apbs` input files generated by `pdb2pqr` solves the PBE twice in order to calculate the solvation energy of a protein, once with a inhomogeneous dielectric constant (protein and solvent have different dielectric constants) and once with a homogeneous dielectric constant (protein and solvent have the same dielectric constant). For visualizing the electrostatic potential of the protein, one only needs to solve the PBE once for the inhomogeneous medium. Thus, you can delete the second part.
- In the end, the solvation energy is calculated. This part should also be deleted.
- Note, the OpenDX file containing the electrostatic potential is written out in the first calculation, i.e. with the inhomogeneous dielectric constant. Give the potential file the same name as your PQR-file to make it easier to associate them.

Task 4: Describe in your protocol the meaning of ”pdie” and ”sdie” using the `apbs` documentation [4]. Which value is set for the solvent radius? What does this radius mean? Which consequence would have a decrease of this value?

Calculate the electrostatic potential of ferredoxin (you need to edit your `apbs`-input file). Set the protein dielectric constant to 4.0. Give the tabulated name to the OpenDX file that will be generated. Make sure that the PQR-file does not include any water molecules.

Run `apbs` by typing

```
apbs your-apbs-input-file.in > your-apbs-output-file.out
```

in the console. The electrostatic calculation will take a moment...

After the calculation is finished (check the .out-file!), you have a DX-file in your directory containing the electrostatic potential.

5.3 Visualization of the Electrostatic Potential of a Biomolecule

`vmd` can be used to visualize the electrostatic potential. Load the PQR- and DX-file of ferredoxin into `vmd`.

Again: Open the PQR-files as PDB because `vmd` fails to read the PQR format of `pdb2pqr`!

Choose (`File -- New molecule...` in the `VMD Main` window and choose `Determine file type -- PDB`). Adjust the molecule to the desired view. Now select the molecule in the `VMD Main` window and load the DX-file into the molecule (by `File -- Load Data Into Molecule`). Do not be surprised, nothing will have changed for the moment.

Alternatively, you can read the file when you start `vmd`:

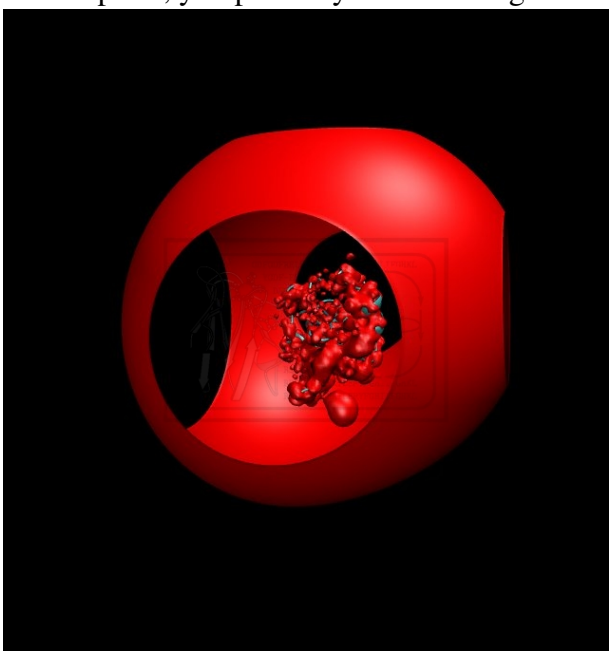
```
vmd -pdb fd.pqr -dx fd.dx
```

5.3.1 Isocontour Surfaces

One of the most popular visualization methods is the **isocontour surface**.

1. First, choose `Graphics -- Representations` from the VMD Main window.
2. Create a useful representation, e.g. cartoon.
3. Hit the `Create Rep` button and change `Drawing Method` to `Isosurface`. The box shows the size of the grid on which the PBE is solved.
4. Change `Draw` from `Points` to `Solid Surface`.
5. Change the `Isovalue` to 3 RT/e (1 RT (298 K) = 2.4789 kJ/mol = 0.5925 kcal/mol, $e = 1.6022 \cdot 10^{-19} C$). Don't forget to press Enter/Return after entering the number!
6. To continue the longstanding tradition of electrostatic potential coloring, choose `ColorID 0` for the `Coloring Method` (positive potentials are colored blue, and negative potentials are colored red).
7. For the negative isocontour, hit `Create Rep` and select the newly created representation. Change the isocontour value to -3 RT/e and the `ColorID` to 1.
8. Change `Show` from `Box + Isosurface` to `Isosurface` to get rid of the frame.

At this point, you probably have an image that looks something like this:



Task 5: Make a figure of ferredoxin showing the positive and negative potential with the isocontour surfaces of -3 RT/e and 3 RT/e and include it in your protocol (white background can be used to save ink). Pay attention that the information you want to convey with the picture is really given (orientation of the molecule, size of the molecule). Why are there holes in the contour (at -3 RT/e, it might be helpful to show the box again)?

5.3.2 Surface Electrostatic Potential

Another popular visualization of electrostatic potentials is to map them to the biomolecular surface. Before proceeding, you may want to delete the two isocontours you just created using the `Delete Rep` button in the `Graphical Representations` window or hide them by double-clicking the representation name in the `Graphical Representations` window.

1. Go to the `Graphical Representations` window (`Graphics -- Representations...` from the `VMD Main` window) and create a new representation of the molecule with the `Create Rep` button.

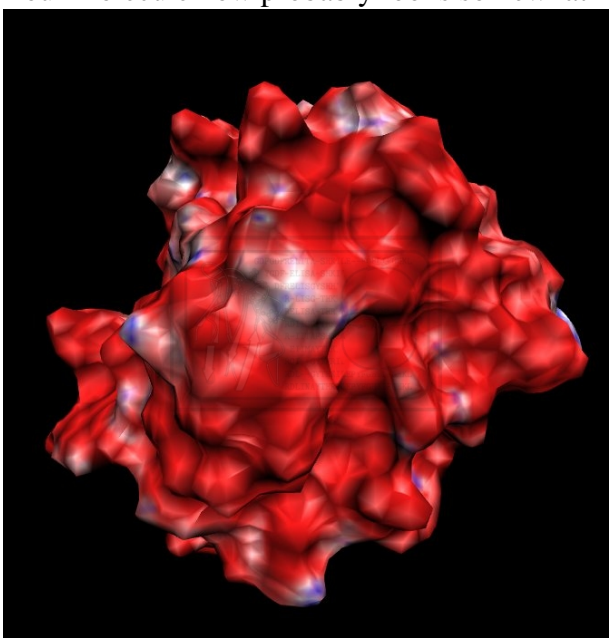
2. Go to the `Draw style` tab of the `Graphical Representations` window and change `Drawing Method` to `MSMS` or `Surf` and `Coloring Method` to `Volume`.

3. Go to the `Trajectory` tab of the `Graphical Representations` window and change the `Color Scale Data Range` to: -10 to 10 (RT/e).

4. Based on your version of `vmd` and your personal preferences, you have to possibility to change the color scale for this image. Go to `Graphics -- Colors...` in the `VMD Main` window and select the `Color Scale` tab from the new `Color Controls`

window. The traditional coloring scheme for electrostatics is "RWB" (in the `Method` menu). Be traditional with the coloring scheme!

Your molecule now probably looks somewhat like this:



Task 6: Include the picture in the protocol! A white background can help you to save ink or toner.

5.3.3 Ionic Strength Dependence

Currently, the calculation of the electrostatic potential of ferredoxin is done without ions in the solvent. In this section we want to analyze the effect of ionic strength.

Change the setting to have 0.1 M NaCl present in the solution (keyword `ion` in the `elec`-statement). Set the ion radius to 2.0 Å. Use the documentation to find out how the settings need to be changed. You need to include two lines, one for Na^+ and one for Cl^-. Give a new name to the DX-file that you will generate (modify the line `write pot dx ...`)

Task 7: Make again two pictures with isocontours at +/- 3 RT/e and surface electrostatic potential at +/- 10 RT/e. Use approximately the same orientation as in task 6 (use the coordinate system in the left lower corner). Compare these picture to the picture obtained from the calculation without salt. What do you observe? Why?

5.3.4 Binding Interfaces

We want to characterize the binding interface of FNR.

Therefore, repeat the calculation with ionic strength for the three other structures you generated (FNR, FNR-NAP, complex). Make sure you use file names according to the table above for all files!

Generate pictures with electrostatic potentials mapped to the surface for each structure, i.e. complex, ferredoxin, FNR and FNR-NAP. **In the figures of FNR, FNR-NAP and ferredoxin, the binding surface should be well visible!**

It is recommended, that you load all four structures (with PQR and DX file) into `vmd` at once. You can switch between the molecules by selecting them in the main area of `VMD Main` and choosing `Molecule -- Toggle Displayed` from the menu. You should not fix a molecule `Molecule -- Toggle Fixed` to ensure that all rotation and translation operations are always done on all molecules (independent if they are currently visible or not).

Task 8: Include the four pictures in the protocol. What can you conclude about the interaction of the FNR with ferredoxin? What can you say about the NADP binding site (with and without NADP bound)?