

# Linux/GAP/emacs Reference Guide

This reference guide contains “technical” details necessary to use GAP for the labs. The first section is by far the most important part of this document.

## 1 Logging On and Logging Off

The brief directions for a GAP session are:

1. Start a Linux session.
2. Move to the appropriate directory and obtain any required files.
3. Start GAP.
4. Read any required files into the GAP session.
5. Log the current session for later editing.
6. Exit GAP.
7. Logoff.

Each of these is explained in more detail below. Some steps only have to be done once - those are the steps containing “unless you have done so previously”. Other commands must be done in every GAP session.

### 1.1 Start a Linux session.

- **Additional steps if the “Welcome to Linux” dialog box does not appear.**
  - Your computer may be running the Linux operating system, but the previous user did not log off. In this case, press the **Ctrl-Alt-Backspace** key sequence.
  - Your computer may be running Windows XP. In this case, press the **Ctrl-Alt-Del** key combination. Select **Shutdown** and then **Restart** from the resulting dialog boxes. Eventually, a black and white screen will appear. From this screen, use the arrow keys to highlight Linux (**not Windows XP**).
- **In the “Welcome to Linux” dialog box, enter your login name and password in the appropriate spaces and click Go!**
- **Open a terminal window.**
  - Open a terminal window by clicking on the button with a shell on it. You can find this button in the lower part of the screen.
  - You may want to change the size of the window using the mouse. To do this, position the mouse pointer on the lower right-hand corner, hold the left button down, and drag the mouse.

- **Change your password, unless you have done so previously.**

- Enter the command

```
passwd
```

and following the on-screen instructions. A password should involve a mixture of letters and digits and should be at least six characters in length. Be sure to remember this new password!

## 1.2 Move to the appropriate directory and obtain any required files.

- **Make a directory for the course files, unless you have done so previously.**

- Make a directory 5100 with the command

```
mkdir 5100
```

- **Change to the course directory.**

- Use the command

```
cd 5100
```

- **Make a directory for the current lab, unless you have done so previously.**

- For Lab 1, for example, you would use the command

```
mkdir lab1
```

- **Change to the current lab directory.**

- As an example, to change to the lab1 directory from the 5100 directory, the command would be

```
cd lab1
```

- Verify that you are in the correct directory with the command

```
pwd
```

For future labs, the output from this command should be

```
/home1/cuxxx/5100/lab1
```

where your login name should appear instead of `cuxxx` and the name of the directory for the current lab should appear instead of `lab1`

- **Obtain any required files, unless you have done so previously.**

- To obtain the files, open a Netscape session by entering

```
netscape &
```

at the command line. In Netscape go to the URL

```
http://www.ux1.eiu.edu/~cfdmb/5100
```

**Right** click on **File Needed** after the name of the current lab and select **Save Link As** from the resulting menu. Click **OK** to download the file.

### 1.3 Start GAP

- Return to the command window and start an GAP session by entering

```
xgap &
```

at the command line.

- It will take a short time for GAP to load.

### 1.4 Read any required files into the GAP session.

- *Each time you start a GAP session to work on a lab, you may need to read in any file needed for that lab.*
- The names of the files for each labs will be given in the laboratory descriptions. No such file is needed for Lab 1. However, if, for example, `GCD.gap` were needed, you would download this file from the web-page and read into the current GAP session with the command

```
Read("GCD.gap");
```

### 1.5 Log the current session for later editing.

- You will want to log the current session to a file for later editing. As an example, the command

```
LogTo("Session1.gap");
```

will log the current session to `Session1.gap`. Warning: If a file of this name already exists it will be overwritten!

- Proceed with the GAP session. When you are finished with the session, close the logging file with the command

```
LogTo();
```

### 1.6 Exit GAP.

- To exit GAP, enter `quit`;
- You can edit your logged files using `emacs`, see §5.
- Be sure to sign off before leaving the lab.

### 1.7 Signing off

- **It is very important to do this every time you are about to leave the lab. Failure to do so leaves your account and files at risk.**
- Click on the **logout** icon. It is an orange icon on the task bar near the bottom of the screen and it is directly below the blue “padlock” icon. In the resulting “End session” dialog box, select **Login as a different user** and then **OK**. This should bring up the “Welcome to Linux” dialog box.

## 2 Linux Details

It will be useful to explain a little (very little) about the Linux operating system.

### 2.1 Basic Commands

The following Linux/Unix commands should be sufficient for most users. (The commands are given in alphabetical order, not the order of importance.)

command	action	example	result
<code>cd</code>	change directory	<code>cd 5100</code>	changes from the current directory down to 5100
<code>emacs</code>	start editor	<code>emacs lab1.gap &amp;</code>	starts the <code>emacs</code> editor in order to edit the file <code>lab1.gap</code>
<code>ls</code>	list contents	<code>ls</code>	displays the names of files contained in current directory
<code>mkdir</code>	make directory	<code>mkdir 5100</code>	creates the directory 5100 as subdirectory of the current directory
<code>pwd</code>	print working directory	<code>pwd</code>	displays the absolute path name of the current directory
<code>startx</code>	start desktop environment	<code>startx</code>	starts the X-Window graphical user interface

### 2.2 Multi-tasking

In Linux, by adding an ampersand to the name of the programs, you can have several programs running at the same time. This feature will be very useful when using GAP. For example, you can open a terminal window and enter

```
xgap &
```

to start a GAP session. Then return to the terminal window and enter

```
netscape &!
```

to start a netscape session. Near the top of this window there will be a line which gives the location of the html file currently being displayed. Change this entry to

```
/home1/apps/gap4r4/doc/htm/index.htm
```

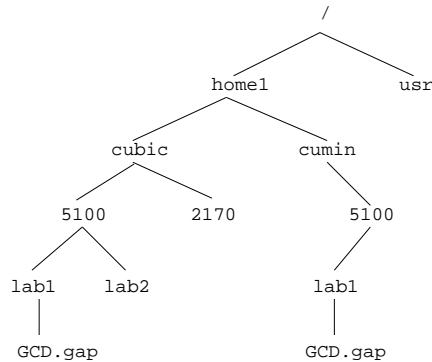
and the GAP index page will be displayed. And finally, return to the terminal window and enter

```
emacs Squares.gap &
```

for editing the file `Squares.gap` using `emacs`. By using the mouse, you can bounce between the various applications.

### 2.3 File Structure

UNIX and Linux use a tree-based directory structure. The following diagram illustrates this structure.



Each file and directory is given an **absolute path name**, as indicated in the tree. The root, indicated by /, contains all files. In the example above, based upon the directory structure of the server used in this course, the root directory has two subdirectories — /usr and /home1. The directory /usr contains the executable programs, like gap and emacs while /home1 contains all user files. The directories /home1/cubic and /home1/cumin contain files of the users cubic and cumin, respectively. cubic's files for Mathematics 5100 are contained in /home1/cubic/5100 and his files for lab1 of the course are contained in /home1/cubic/5100/lab1.

Absolute path names all start with the root directory and can become quite long. **Relative path names** provide a much shorter notation and are based upon the current working directory, whatever that might be. A single dot represents the current working directory and a double dot represents the directory one step closer to the root. A tilde (~) represents the user's home directory. The following table indicates various relative path names for the user cumin when the current working directory is /home1/cumin/5100/lab1

Current working directory: /home1/cumin/5100/lab1	
absolute path name	relative path name
/home1/cumin/5100/lab1	.
/home1/cumin/5100	..
/home1/cumin/5100	../.
/home1/cumin/5100/lab1/GCD.gap	GCD.gap
/home1/cumin/5100/lab1/GCD.gap	./GCD.gap
/home/cumin	~

### 3 GAP Fundamentals

GAP (for Groups, Algorithms, and Programming), a powerful package for doing computational discrete algebra, was developed in Aachen, Germany and is now maintained at the University of St. Andrews, in Scotland. This package, started in 1985, currently consists of a self-contained programming language, over 800 built-in functions for the study of algebraic structures, and over 1000 pages of documentation. XGAP runs on top of GAP and allows one to use X-Windows. GAP is distributed free of charge to the mathematical community, see

[http://www.\\_groups.dcs.st-andrews.ac.uk/~gap/](http://www._groups.dcs.st-andrews.ac.uk/~gap/)

for further details.

If you would like to install it on a machine that you have, I will be willing to help. GAP is available for PC's running a 32-bit operating system (Like Windows 95 or later), for MacIntosh's running Mac OS, and for any Unix platform. XGAP, which will be used in this course, is available only under a Unix platform.

- The up arrow key can be used to recall and edit previously entered commands. In fact, if you enter, say for example `Re`, and then press the up arrow key, the last command, if any, which started with `Re` will be recalled. This capability greatly simplifies typing.
- Almost every line in `GAP` must end with a semicolon.
- In `GAP`, the symbol `:=` is used for assignment, the symbol `=` being reserved for the relation of equality.
- The basic syntax of `GAP` is much like that of `C++` (or any other computer language). One major difference between `C++` and `GAP` is that in `GAP` not all objects need be declared before they are used.
- The standard `GAP` prompt is `gap>`. However, if you are entering a multi-line command, like a loop or a function definition, the prompt becomes `>`. Sometimes, an error will occur and the prompt will change to `brk>`. From the `>` or the `brk>` prompt, you can return to the `gap>` prompt by pressing the `Ctrl` and the `D` keys simultaneously, (i.e. by entering `Ctrl-D`). **However, entering `Ctrl-D` from the `gap>` prompt will cause you to exit `GAP`, so be careful.**
- The complete manual for `GAP` is available in HTML and can be read with any browser. For details, see §2.2

## 4 Programming in GAP

The programming capabilities of `GAP` will be more fully described in the laboratory descriptions, but here is a summary of some of the more important features.

- A sequence of steps can easily be repeated by using a “for loop”. For example, the following `GAP` session will
  - Create a storage space and initialize it to zero.
  - Store the sum of the integers from 1 to 20 in that space.

```
S := 0;
for k in [1..20] do
  S := S + k;
od;
```

Notes:

1. Observe that there is no semicolon after `do`
  2. Observe that the end of the loop is marked by `od`
  3. This loop will not produce any output. Enter `S;` on a line by itself to see the value that was computed.
- It is possible selectively execute statements based upon the value of a variable. The following `GAP` session will
    - Create a storage space and initialize it to zero.

- Store the sum of the even integers from 1 to 20 in that space.

```
S := 0;
for k in [1..20] do
  if (k mod 2 = 0) then
    S := S + k;
  fi;
od;
```

Notes:

1. Observe that there is no semicolon after **do** or **then**
  2. Observe that the end of the **if** statement is marked by **fi**
  3. Observe that **=** denotes the equality relation and **:=** is used for assignment.
  4. This loop will not produce any output. Enter **S;** on a line by itself to see the value that was computed.
- There are many functions available in **GAP** and you can write your own. In writing such a function, it is necessary to specify the name of the function; local variables (if any); and the object to be returned by the function (if any). Local variables are those that only have meaning within the function, while the variables which have meaning within the main **GAP** session are called global variables. Variables which are not specified as local are assumed to be global. Thus, an error message will result if a variable which is neither previously defined nor declared as local is used in a function. As an example, consider the following function which finds the sum of squares of the first  $n$  positive integers.

```
gap> SquareSum := function( n )
> # Purpose: To find the sum of the first n squares
> # Input: n - positive integer
> # Output: S - the sum of the squares
> local k, S;
> S := 0;
> for k in [1..n] do
>   S := S + k^2;
> od;
> return S;
> end;
```

Notes:

1. The lines which begin with **#** are comments.
  2. Observe that there is no semicolon after the first line.
  3. Observe that the function terminates with **end;**
  4. To invoke the function to find, say, the the sum of the first 10 squares, enter **SquareSum(10);**
- You can write the **GAP** commands for your session in another file and read these into **GAP** . This simplifies typing somewhat and allows easier error correction. Suppose you wanted to create a file, **SquareSum.gap**, containing the function described above.

- Leave the GAP session active. Return to the terminal window and start an `emacs` session to edit the file. In this case, the command would be

```
emacs SquareSum.gap &
```

- Enter the definition of the function in this file and save it. (See §5, below, if needed.)
- Return to the GAP session and read in the file by entering

```
Read("SquareSum.gap");
```

Notes:

1. If you enter commands using an `emacs` session and then read the file into a GAP session, the only output to the screen will be that which results from a `Print` command. If you enter the commands directly into GAP there will be output from virtually every command.
2. Using an `emacs` session is excellent if you want to define a function (which requires several lines of input), but for simpler commands it is easier to just enter them directly in a GAP session.

## 5 emacs Fundamentals

The `emacs` editor provides a way to edit files. For example, to edit the file `session1.gap` in the current working directory, issue the command

```
emacs session1.gap &
```

There are pull down menus which give you various options.

- To print the file you are editing, select `Tools`, then `Print`, then `Postscript Print Buffer`.
- To save the file you are working on, select `Files`, then `Save Buffer`.
- To exit `emacs`, select `Files`, then `Exit Emacs`