
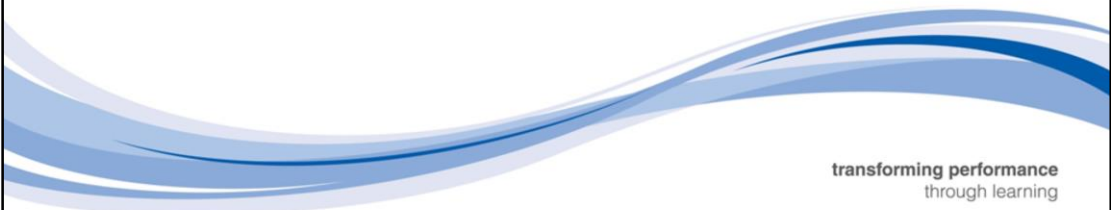


QACADV_v1.0



Appendix - Introduction to Linux

Advanced C



transforming performance
through learning

Linux for C Programmers

- **Objectives**
 - To be able to use Linux to create, compile, and test C programs
- **Contents**
 - The Desktop
 - The Command Line
 - The HOME Directory
 - File Names and Directories
 - Basic Commands
 - Text Editors
 - Compilation & Link
 - Help!
- **Summary**



This chapter is not intended to teach you Linux! It is designed to teach you enough to run the exercises for this course, nothing more.

The Linux distribution used will either be Red Hat 9 or a version of Fedora. So far as this course is concerned they are very similar.

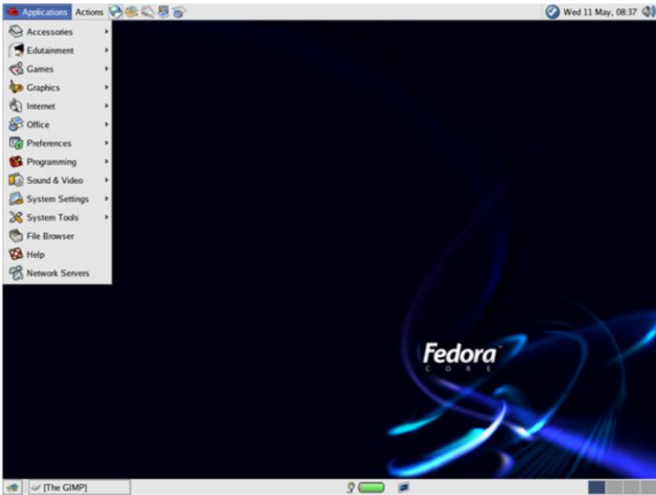
QACADV_v1.0

The Desktop

- **Linux supports a variety of different desktops**
 - Default with Red Hat and CentOS Linux is called GNOME

start button

Open Office products



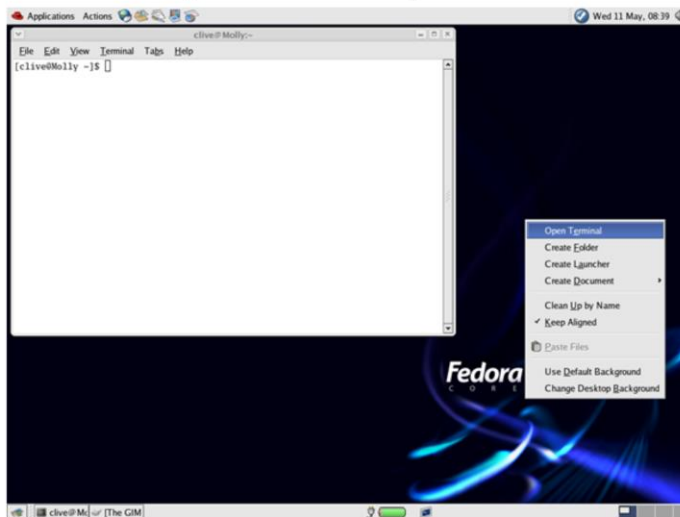
The screenshot shows the Fedora Linux desktop environment. The desktop background is dark blue with the 'Fedora' logo and a blue abstract design. The top panel contains the 'Applications' menu, which is currently open, displaying a list of application categories such as Accessories, Education, Games, Graphics, Internet, Office, Preferences, Programming, Sound & Video, System Settings, System Tools, File Browser, Help, and Network Servers. A red arrow points from the text 'start button' to the 'Applications' menu icon. Another red bracket points from the text 'Open Office products' to the 'Office' category in the menu. The bottom panel shows a taskbar with a single window titled '(The GIMP)'.

The Linux Desktop is fairly intuitive, as most GUI's are. The default in this version of Linux is called 'Blue Curve', which is a version of Gnome. If you are familiar with Linux and prefer, for example, KDE, then feel free to use this instead. Please remember, however, the aims of this course!

Clicking on the Red Hat icon on the menu bar will produce the main GUI menu, from here it is fairly simple to navigate to sub-menus. You will not be using any of these applications today.

The Command Line (Shell)

- **Right-mouse click, then choose *Open Terminal***



- **We will be using the Korn Shell**

Most of our interaction with the operating system will be via C programs, but we still need to create the source code and compile it. We might also need to perform some basic file housekeeping, and the command line interface, called the *shell* on Unix systems, is a simple interface to use.

To invoke a shell terminal session on Fedora, right click on the blue background and select 'Open Terminal' (on Red Hat 9 the menu item is called 'New Terminal'). This should start a window like the one shown above. The prompt displays the *current working directory*.

In your terminal session you might find that some characters are not displayed correctly. If so, select Terminal from the menu bar, 'Set Character Encoding', then 'Add or Remove', select Western (ISO-8859-1) and 'Add'. Now go back to the Terminal menu, Set Character Encoding, and select Western (ISO-8859-1).

Each user on Unix has a default shell used at login time. For this course we have set it to the *Korn shell* (/bin/ksh), which is a popular shell from the Unix world. Gaining in popularity is the *Bourne Again Shell* (/bin/bash) which is normally the default on Linux. The *C Shell* (/bin/csh) was intended for use by C programmers but requires a rather steep learning curve, so we suggest you stay away from that unless you are already familiar with it.

To run the Bourne Again Shell, in a terminal session type **bash**.

To run the C Shell, in a terminal session type **csh**. This is actually the public domain "Turbo C Shell", **tcsh**.

To alter your default shell permanently use the "change shell" command, **chsh**. For example to alter your user to *always* use bash:

```
chsh -s /bin/bash
```

You will be prompted for your password, and the shell will be changed next time you logon.

The HOME Directory

- **Every user on the system has a “home directory”**
 - It is the location the user is put in when he first logs into the system
 - Becomes the working (current) directory at login time
- **Contains user's own files**
 - User can freely create files and directories here
 - Home directory contents is normally protected from other users
- **Holds user-specific configuration files**
 - Not normally seen, file names are prefixed with a . (*dot*)
- **Default directory for the `cd` command**

Every user has a 'home directory', which has a number of uses. It is the current directory when a user signs in to an interactive shell, and it stores user specific configuration files. The names of the configuration files depend on the applications in use. Normally a user's files will reside in their home directory, but any other directory may be used provided the user has sufficient permissions.

In this course you can make any changes you wish to your home directory, including creating directories (**mkdir**).

File Names and Directories

- **Filename**s not starting with / are relative to the *current working directory*
- **To change the current working directory:**
 - use the shell command `cd`

<code>cd directory-name</code>	move to <i>directory-name</i>
<code>cd ..</code>	move to the <i>parent</i> directory
<code>cd</code>	move to the user's <i>home</i> directory
- **The shell carries out *globbing* (wildcards)**

<code>*</code>	zero or more characters
<code>?</code>	exactly one character
<code>[abc]</code>	one character, either a or b or c
<code>[!abc]</code>	one character, neither a nor b nor c
<code>~</code>	The user's home directory

<code>cd q*v</code>	move to the directory that starts with q and ends with v
---------------------	--

A file or directory name can be given using an *absolute path name*, or a name relative to the *current working directory*.

An *absolute path name* is unique on a given system, and must start with a / (forward-slash). On Unix and Linux there is only one root directory, named / (whereas on Microsoft Windows there is one on each partition).

Every process on Linux (as well as Unix and Windows) has a *current working directory*, which is usually inherited from its parent. File or directory names not prefixed with / are assumed to be found relative to the current working directory. Normally this directory name is displayed in your prompt at the command line, but if you are not sure then just type **pwd** (print working directory).

There are a number of shortcuts for directory names. By default an "undecorated" file name will be assumed to reside in the current directory, and a single 'dot' can also be used to refer to the current directory. The current directory's parent can be referred to by two dots.

Both . and .. can be used as qualifiers to file or directory names used in C programs, since they actually exist as entries in the directory. Shortcuts using globbing (also known as wildcards) cannot be used directly inside a C program, since the shell expands these for us. Note that this includes the tilde character ~, which is a shortcut to the user's home directory.

The GNU C runtime includes libraries and routines for carrying out globbing from C, using a function called (predictably) **glob**. This routine is non-standard and outside the scope of this course.

Basic Commands

- **Rule 1 – Everything is case sensitive – *just like in C!***
- **Listing files**
 - `ls filespec`
 - `ls -l filespec` *long listing*
- **Copying a file**
 - `cp from to`
- **Removing a file**
 - Warning! When it is done, it is done! There is NO "undelete"!
 - `rm file/s`
- **Executing a program**
 - Type the program name, followed by any arguments (argv)
 - Uses the environment variable PATH to find it (and nothing else)

Commands and file names are case sensitive on Unix and Linux systems. This should not be a surprise to a C programmer!

There are a huge number of commands available on Linux, actually most are programs written in C. Fortunately you only need to know a few to be able to do useful things. Those shown on the slide are probably more than you will need for this course.

If you are not happy with using the command line, then you may notice an icon on the desktop called "User's home". This is the Nautilus file manager, which is similar to Microsoft's Windows Explorer. Many file operations can be carried out using this instead of the command line.

Text Editors

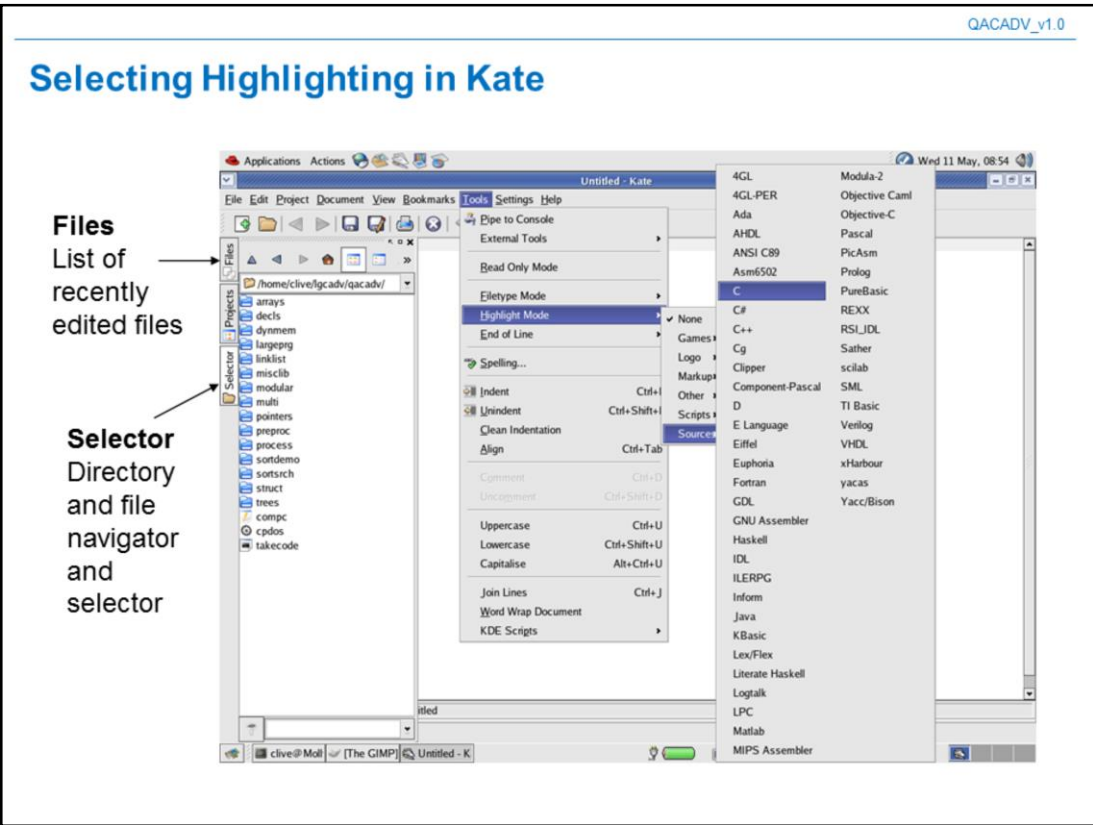
- **A number of graphical editors are available:**
 - kate
 - nedit
 - Run from the shell in the background like this:

kate &
- **For Unix experts:**
 - vi
 - The Linux version is vim (Vi IMproved)
 - Includes context highlighting
 - emacs (and Xemacs)
 - Very powerful development tool
 - Kdevelop
 - Heavyweight Integrated Development Tool

Which is best?
That depends on
your own personal
preference!
We recommend **kate**
for the uninitiated

Linux comes bundled with a range of editors, and there are many more available on the Web. The editor chosen by a programmer is a personal choice, and comparisons are subjective. We don't mind which text editor you use, many Unix programmer's will be happy with vi or emacs. If you are new to Unix or Linux then try **kate** or **nedit**, they are simple to use and have context highlighting for C programs. Following slides will show you how to use them.

There are also debuggers available, but require practice to use. The GNU debugger is call **gdb**, and can be driven from the command line or through the GUI utility **ddd** (Data Display Debugger). The editor used by ddd is **vi**, so we cannot recommend it to newcomers.



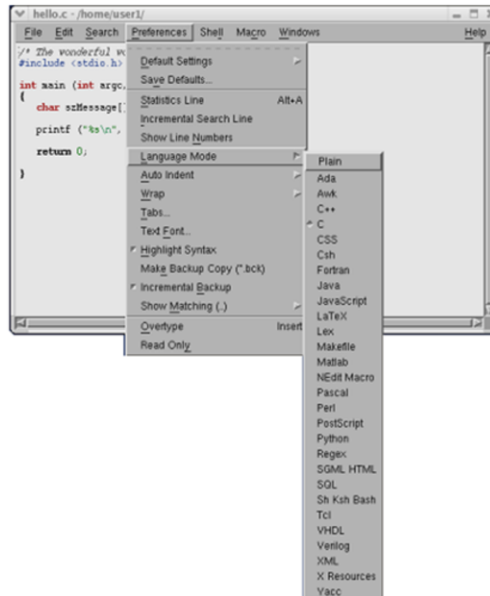
Kate is part of the K-Office suite of programs, and is based on Kwrite which in turn is part of the KDE project. The version of **kate** shown is that distributed with Fedora Core 3 (the Red Hat 9 version is slightly different). The panels presented by default are the main edit panel, with tabs on the left-hand side. The tab of most interest is the Selector, which enables easy navigation between directories (folders). There are many other features, but none are essential for this course (but feel free to explore).

There is no integration with the compiler or debugger at present. Kate's main disadvantage is the system resource required to run it, you will probably notice a pause while it starts.

Selecting Highlighting in Nedit

Most editors have a context sensitive highlight mode.

This gives a hint to miss-spelt keywords and variable types



Nedit does not rely on KDE, which can be heavy on system resource. It does not have quite as many features as Kate, but has enough for most tasks. Like Kate, there are no links to the compiler or debuggers.

Compilation & Link

- **The C (and C++) compiler: gcc**

gcc [-std=*standard*] [-g] [-O*level*] [-I*dir...*] [-o *outfile*] *infile*

- **Usual options are minimal**

gcc -o hello hello.c

- **Optimisation can be controlled by -O*level***

- O1 basic optimisations for memory and speed
- O2 O1, plus most optional optimisations
- O3 O2, plus inline functions and register renaming
- Os Optimise for size



- **Compile for debug using option -g**

- **Set preprocessor macros with -D*name***

Oh,
not zero

The *de facto* compiler used on open source systems is the GNU compiler, **gcc**. It is produced and maintained under the umbrella of the Free Software Foundation, which has hundreds of volunteers working on its products. Many software houses and hardware manufacturers also participate.

The compiler has a huge range of options, but most are not needed. The **-o** option gives the name of the executable output file. Don't expect a **.exe** extension, Unix/Linux does not use that. If you forget the **-o** option then the default output file name is **a.out**, for historical reasons.

The **-std** option can be any of the C standards. To make life easier there is a script named **c99** which calls **gcc** with **-std=iso9899:1999** (call the **c99** script as you would the compiler).

The **-O** option, for optimisations, is shown because it is required for some exercises in the Advanced C course.

A program can be compiled in debug mode using the option **-g**. Unlike some compilers, with **gcc** you can still use optimisations with debug, but this can give weird results during debugging. The compiler does not set any special macros in debug mode, but you can set them yourself with the **-D** option, for example **-DDEBUG**. This can be tested in the program with **#ifdef DEBUG**.

Calling the compiler invokes the linker automatically, which will include the C runtime library by default. Sometimes additional libraries are required, for example to include the maths library append the option **-lm**.

Calls to the compiler are often bundled into a script file used by the **make** utility, usually called "Makefile". Makefiles have been created for the exercises, so you might not need to use the compiler directly in this course.

Help!

- **Linux comes with full on-line documentation**
 - Known as 'man pages'
- **Use the manual pages to:**
 - Get a description of a command or library function: `man function`
 - Locate a function using keyword: `man -k keyword`
- **Organisation is by section**
 - Section (1) – command line programs
 - Section (2) – operating system (kernel) calls
 - Section (3) – C library functions
- **Refer to a standard name using the section number**
 - `write(1)` is very different to `write(2)`

`man 3 fopen` *defaults to lowest section number*
- **The program used to display man pages is `less(1)`**
 - navigate using <PgUp>, <PgDn>, arrow keys
 - <q> to quit
 - search using */Regular-Expression*

The manual pages, better known as 'man' pages, provided the best reference to Linux commands and C library functions. You may need to refer to the manual pages to determine how a command behaves and what it is used for.

All manual pages are organised in a similar way, with key paragraphs clearly marked. The **-k** option allows you to search database of manual pages for a keyword.

The sections of most interest to C programmers are 2 and 3. Section 2 functions are system specific, and usually require and at least one additional header file for the prototype (like `unistd.h`). Section 3 (on Linux) is further subdivided:

(3C) These functions, the functions from chapter 2 and from chapter 3S are contained in the C standard library `libc`, which will be used by `cc(1)` by default.

(3S) These functions are parts of the `stdio(3)` library. They are contained in the standard C library `libc`.

(3M) These functions are contained in the arithmetic library `libm`. These require the option `-lm` (ell-em) to the C compiler (`cc(1)`).

Other sections exist, but are not relevant to this course. You can specify the section number as a second argument to the `man` program, for example: `man 2 write`. Usually you do not need to do that, but if there happens to be a library function with the same name as a Unix command line program, then it will pick up the section 1 documentation by default.

Note: Often GUI programs do not have man pages. For example, `kate` does not, yet `nedit(1)` does!

Summary

- **Each delegate has a separate Linux system**
- **The command line interface is known as a "shell"**
 - We are using Korn shell, but you can change it
- **Only a minimal knowledge is required for this course**
 - This is a C course!
- **Many editors are bundled with Linux**
 - Use whichever suites your development style
- **We will be using the GNU C compiler**
 - scripts will be provided
- **Get help on library functions with `man function-name`**

