INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

The United Nations University

SMR/774 - 3

THIRD COLLEGE ON MICROPROCESSOR-BASED REAL-TIME CONTROL - PRINCIPLES AND APPLICATIONS IN PHYSICS
26 September - 21 October 1994

PRACTICAL LINUX

Catharinus VERKERK
Computing and Networks Division
CERN
CH-1211 Geneva
Switzerland

Microprocessor-based Real-Time Systems.

ICTP,Trieste,

26 September-21 October 1994

Practical Linux

C. Verkerk,

CN Division,

CERN,

CH1211-Geneva 23.

- Linux is a UNIX clone for PCs, freely available.
- You may install it, use it and even distribute it, provided you obey the "Copyleft Statement"
- You may also modify it, as long as you clearly mark your modifications, so that the original authors are not held responsible for your errors.
- It is a very complete system, with many packages and utilities you find on UNIX systems.
- For instance:
  - Languages: C, C++, Objective C, Pascal, Fortran, CLisp,
  - Editors: emacs, jove, joe, vi,
  - Debugger: gdb,
  - the usual UNIX tools: make, sed, grep, etc.,
  - X11, XView, and many widgets,
  - TeX and LaTeX,
  - TCP/IP, FTP, e-mail packages,
  - News readers,
  - Administrators tools: Tk/TCL, Perl,
  - information and documentation: man-pages, but also HOWTOs, Manuals, FAQs.

- These lectures will introduce you to the system and its capabilities, from a practical, users' point of view.
- We will avoid entering into details of the internal workings of Linux. The lectures on "Principles of Real-Time Operating Systems" will delve into characteristics of real-time systems, using Linux for the examples, whenever appropriate.
- Before starting to work with Linux, you must lose certain bad habits (from MS-DOS):
  - *Golden Rule No. 1:* NEVER switch off your PC, NEVER!!
  - *Golden Rule No. 2:* NEVER push reset on your PC, NEVER!!
  - *Golden Rule No. 3:* NEVER hit ctrl-alt-del at the same time, NEVER!!

You need an instructor to SHUTDOWN your machine. You will see later why.

Linux is a new implementation of UNIX, without a single line of the original code, written by a community of "hackers", led by Linus Thorvald. It is distributed on the same conditions as software from the *"Free Software Foundation"*

GNU, from the *"Free Software Foundation"* , also runs on Linux and is the source of many of Linux's packages.

Why is UNIX so popular?

- UNIX introduced a new philosophy in operating systems.
- It presented this philosophy in a coherent manner.
- It proved to be relatively simple to enhance and adapt to new developments, such as networking, distributed systems, distributed file systems, etc.

A UNIX system has a kernel, which is in direct "contact" with the hardware and which provides services to the user programs.

The standard commands, the compilers, editors, text processing programs, etc., all are user programs.

Many pre-UNIX operating systems were monolithic, in the sense that nearly everything (*command interpreter, utility commands, compilers, assemblers, etc.*) were an integral part of the operating system itself.

In UNIX, a user program accesses the kernel services via a system call. More on this later.

The command interpreter is also a user program. So, two different users can use different shells, or even a single user may choose to change to another shell.

I implied that UNIX is a multi-user, multi-tasking operating system. There may be several users logged in to the machine, and each user may run several tasks or programs simultaneously.

This is also what you want to have for a **real-time system**, for reasons we'll see again later. However, this does not necessarily mean that UNIX or Linux can be considered to be true real-time systems.

UNIX also introduced a **hierarchical file system**, with **directories** and **subdirectories** in a **tree structure**. This has been copied to all operating systems developed since.

A **physical device** (hard or floppy disk for instance) can contain more than one **"filesystem"**, built up from a **root directory**. Each of such a filesystem can be mounted on the base filesystem, thus making all files accessible, even if residing on several physical devices (which may have different characteristics).

Another fundamental innovation of UNIX has been that **Input/Output devices** are **treated** just **as ordinary files.**

Normally, a program running under UNIX gets its **input** from a device (or **path**) called **standard input**. Its **output** goes to **standard output** and **error messages** are written to **standard error**. **stdin** is usually the *keyboard*, **stdout and stderr** are usually the *screen*.

stdin, stdout and stderr can be **re-directed to a file.** This was another important innovation introduced by UNIX, copied now by everyone.

Moreover, **stdout** of one program can be **connected to stdin** of another program. The two programs run concurrently and are said to form a **pipe.** The stdout of the second program can be piped into stdin of a third program, etc. This leads to the notion of a **filter program.**

This illustrates another important philosophy of UNIX, namely that programmers should be able to **do complicated things by judiciously using simpler programs** and combining them.

stdin of a program can be **re-directed from a file.** If the program is the **shell**, then this means that the shell can execute a previously prepared **script.** Combined with facilities built-in in the shell, such as *loops* and *if-then-else*, this makes it possible to do very complicated things repeatedly, without the need to type each time long series of commands.

UNIX was nearly entirely written in a **high-level language (C)**, which was also revolutionary.

We now turn to **Linux**, which **behaves** for all practical purposes **as UNIX.** We will have a look at what we find on our machines.

I. Files and File Manipulation

Starting from the **root of the filesystem**, we find a number of **directories**. The important ones are:

bin	**home**	**root**
boot	**lib**	**sbin**
dev	**lost+found**	**usr**
dos	**mnt**	**tmp**
etc	**proc**	**var**

In **/home** we find **subdirectories:**

bozo	**team**
part1	**rinus**
part2	

When I log on to the system with my ID: **"rinus"**, I'll find myself in a working **directory:**

/home/rinus

I have a subdirectory there, called **"tmp"**. From my working directory I can access a file in the directory tmp with: **`tmp/filename`**
Full path: **`/home/rinus/tmp/filename`**

I can change the working directory: `cd tmp`
or `cd /home/rinus/tmp` using the **relative path** or **full path** respectively.

After changing directory I can access the file /home/rinus/myfile as:

`../myfile`

and the file filename either as: `filename`
or as `./filename`

.. means this **current directory**
.. means the **parent** of the current **directory**.

You can see the names of the files in a directory with `ls`. ls allows many options:

`ls -l` shows, among other things, also the **access rights.**

Access rights are in the form of three letters:

r for **read**
w for **write.**
x for **execute**

There are **three groups of three letters**, for the **owner** of the file, for the **group** he belongs to and for the **entire world.**
Thus: **..rw.rw...**

The owner and his group are allowed to read from and to write to the file in this particular case.

Directories have similar **access rights.**

What will happen if from /home/rinus/tmp I do:

`cd ../../part1 ?`

If I were the **superuser** (called "**root**" on all systems) I could do this. The superuser can do everything he likes. A **very dangerous** person! He can kill your job, invalidate your password, delete all your files (and my files as well) etc, etc.

The **access rights protect** you against abuse by other users, but not against the superuser.

For **UNIX** (and, of course, **Linux**) all files are just a **collection of bytes.** The systems does not distinguis between text files, executable files or bitmaps or what have you.

In fact, all files are stored on disk in an identical manner. But the system will complain if you try to execute, for instance, a bitmap.

There are several commands to manipulate files and directories:

`cat`	lists a file on the screen
`more (or less)`	lists a file on the screen, page by page
`cp`	copies a file
`mv`	changes the name of a file, or moves it from one directory to another
`od`	makes an octal dump of a file
`rm`	deletes (removes) a file
`chown`	changes the owner of a file
`chmod`	changes the access rights
`mkdir`	creates a new subdirectory
`rmdir`	deletes an empty directory
`cd`	changes the working directory
`ls`	lists the contents of a directory

For examples, see the exercises.

All files and directories you have seen so far are in the same **filesystem**, which occupies one partition of the hard disk.

Linux can access the **DOS partition** of the hard disk. The "DOS filesystem" is in fact "**mounted**" on the base filesystem. Try:

`ls -l /dos`
or `ls -c /dos`

You can manipulate files in /dos just as Linux files, but you **cannot execute** them. Be aware of the LF vs CR-LF problem and of restrictions on filenames in DOS.

A floppy disk also contains a file system of its own

To access files on a floppy, you must **mount** its filesystem on a **mount point** of the base filesystem:

`mount -t msdos /dev/fd0 /mnt`

When you have finished, you must **unmount** it.

Golden Rule no. 4:
NEVER pop a floppy out of the drive, NEVER!!

Only the superuser is allowed to mount and unmount floppies, which is logical when you think about it.

However, this is not as bad as it looks: Linux has a complete **set of commands** to deal with **MSDOS files**; they even have the names they have in DOS, preceded by the letter **"m"**:
mdir, mcopy, mdel, etc. In addition, they accept **MS-DOS pathnames** (with "\" or "/", at your choice).

```
mcopy thisfile a:
```

copies the file "thisfile" from the current directory to the floppy.

```
mcopy c:/tmp/junk ..
```

copies the file "junk" from the DOS partition of the hard disk to the parent of the current directory. The transformation LF<-->CR-LF is done automagically.

II Virtual Screens

You have more than one **"virtual screen"** at your disposal. Change to another with **alt-Fk.**

In the new screen you can **log in again** with your ID, or as rinus, or as root (provided you know the passwords, of course).

III Processes

The command **ps** will show what is going on on the system. **ps -a** gives more details.

The superuser can see even more .....

IV Man pages

The so-called **man pages** (from "manual") provide **on-line help.** Just type: **man <commandname>** where "commandname" is the name of the command you want to get help information on.

When you use a command incorrectly, you often get a so-called **"usage message".**

Both man pages and usage messages use a peculiar, but well-defined **notation.**

Format: **command options arguments**

Options and optional arguments are enclosed in [ ]
Compulsory arguments are enclosed in < >
Arguments are separated by **spaces or commas.**
..... means you may specify **more arguments** of the same type.
A list of arguments to choose from is enclosed in {}

V Shell

The original shell was the **Bourne shell:** **sh**

It was followed by others: **C shell:** **csh**
Korn shell: **ksh**
Born again shell: **bash**

On our Linux we can choose between: **sh, ksh, tcsh** and **bash**. **bash** is the default.

All shells have a number of **built-in commands: cd, pwd** and others such as **while, if, else, endif.**

The shell also uses **environment variables.** An important one is PATH. (Environment variables are traditionally written in capitals.)

PATH defines a list of **pathnames of directories to search for executable programs.**

Look at your .profile file or the file /etc/profile to see examples of the definition of PATH and other environment variables.

After booting the machine and logging in, the **default shell** will start running and produce a **prompt** on the screen, indicating that it is ready to receive an order.

When you type a command, for instance

```
cat dog
```

then the shell will arrange for the program **cat** to be loaded and run. It will pass on to **cat** the information that the file to be listed is **dog** (in the current directory).

You may include special characters in the command you type, modifying the meaning of the command.

For instance `cat fish >goldfish`

will not list the file **fish** to the screen, but it will put the **output from cat in the file goldfish** (effectively creating a copy of "fish"). This is an example of **output redirection.**

Input redirection: `wc <text.txt`
(**wc** alone would count the number of lines, words and characters you input from the keyboard, up to the first **EOF character** (which is **ctrl-D**).

2> redirects the standard error output.

Careful with > and 2> if the file exists!

>> and **2>>** will **append output** to a file.

What does **>** do when the file exists already?

There are other important special characters recognised by the shell:

- **#** starts a **comment** if it is the first character on a line. The line is ignored by the shell.
- **;** **separates two commands**, which will be executed one after the other.
- **(** and **)** **group commands** together.
- **|** indicates a **pipe.**

Example: `cat fish | wc >fishcount`
What will happen here?

When the shell receives a command such as

```
cat fish >goldfish
```

it will **fork** and **exec cat**, which then becomes a **child process, spawned** by the shell.

In the case of `cat fish >goldfish`
the shell will **wait till the child dies** (e.g. till **cat exits** when it finishes its job).

If I had typed:

```
cat fish >goldfish &
```

then shell would not **wait** after spawning the child process. It would **return immediately** and give its prompt. I can then type a **new command,** which will be executed while **cat** is doing its job in the **background.**

A **shell script** is a text file, containing a **series of commands** to be executed in succession. It may contain **loop and/or conditional execution constructs,** which will then be **interpreted** by the shell itself.

When you ask shell to interpret a script (simply by typing its pathname), then in reality it will **fork a new instance of the shell.**

For this new shell, the **input is redirected** to the file containing the script. Things you now do in the script (`cd` for instance!) will not affect the original shell.

You can find plenty of examples of scripts on the system (look for instance in **/etc/rc.d**).

Scripts are extremely handy and were yet another innovation of UNIX. However, it is not easy to get yourself into the habit of writing scripts. But, when you have something **lengthy or tedious** to do, you should use a script. It is **worth the effort** in most cases, in particular if the tedious task has to be repeated more than once.

VI - Editors

Several editors are available under Linux:

- **. vi** the **classical UNIX editor**. Very good for experienced users.
- **. emacs** **very complete**. Two versions are available: **one runs under X11, the other does not.** The X11 version allows you to have various windows open and do debugging and editing at the same time. It has extensive help facilities.
- **. jove** (John's own version of emacs) is a **simplified emacs**.
- **. joe** a **simple, but sufficiently powerful editor**. It has a help facility.

You invoke joe with **`joe filename`** and you start typing (the note "New File" automatically disappears). Just continue typing.

If you get stuck do **`^k h`**
This means: hold down the "control" key and hit k, then release "control" and type h.

Choose from the menu at the bottom "**Basics**", or "**Windows**" or whatever, using the cursor and hit "**`Enter`**".

You get rid of the help window with **`^k h`**
`^k x` **saves your file and exits from joe.**

VII - Language Processors

The C Compiler is called **gcc**. It is in fact the GNU **C compiler**.

gcc has literally **hundreds of options**. A number of them are used by default, so don't bother.

In the best of UNIX tradition, **gcc will produce an executable file in the current directory, called a.out**. If you don't like this, either rename (mv) the file, or use the **-o option** of gcc:

```
gcc -o myfile myfile.c
```

The **-g option**, which produces information for the **debugger**, is the default.

gcc will also compile programs written in C++ or in **Objective C**. It detects the nature of the language from the **extension** of the filename:
.cpp for C++, **.cobj** for **Objective** C.

Translators are available for other languages:

f2c	**Fortran to C** translator
p2c	**Pascal to C** translator.

Other languages on the system are: **CLisp** (Common Lisp), **Perl**, and **Tk/TCL**.

There is certainly also an **assembler: gas**

VIII - Network

Our machines are connected to an **Ethernet segment**, which is in turn connected to other Ethernet segments and to the **Computer Centre** of ICTP.

ICTP in turn **is connected to the Internet**.

In principle, packages such as **telnet**, **ftp**, and **electronic mail** (**pine**, **smail**, etc.) should therefore be usable, if they are configured correctly.

IX - Documentation and Text Processing

Linux comes with a **vast amount of documentation** which is not necessarily all available on our machines.

Some of the documents are in a format which can be handled by **groff** (The GNU equivalent of troff or nroff) on the one hand and by **LaTex** on the other hand.

There is also a large amount of **HOWTOs: Ethernet HOWTO, Installation HOWTO, Keyboard HOWTO,** etc.

Also many **"Frequently asked Questions"** find answers in the **FAQ file(s)**.

Then there exists a general **Linux Guide**, a **Network Administrator's Guide** and a **Kernel Hacker's Guide**.

All these can be formatted with **LaTex** and **dvips**. The resulting **postscript file** can be viewed with **Ghostview**.

Also the equivalent of the classical **UNIX text formatter** exists (**groff**).

I leave it to you to discover what is installed on our machines and what is not.

X - UNIX Tools

The UNIX Toolset is also available under Linux. Source code for the kernel and for most of the packages is available (although not necessarily part of the distribution of Linux).

To recompile something you will therefore use make in most cases. It is there, and also the Makefiles which are needed to compile the sources.

Other useful tools are also available:

sed	the stream editor
m4	macro processor
yacc	yet another compiler compiler
grep	for string search
find	to find files
tar	archive facility
gzip, gunzip	file compression/de-compression.

XI - make

make is a command generator.

From a description file and general templates it creates commands for the shell.

make sorts out dependencies among files. A program often consists of several source files, header files, libraries. When one of these is modified, you must rebuild the program, by re-compiling some of the files, but not necessarily all, and then re-linking the object files.

make decides what must be done, basing itself on the dependencies and the last modification date/time of each file.

If file A depends on file B and B was modified after A, then A must be "remade": compiled, linked, edited, substituted in a library, or what have you.

Generally you invoke make by typing:

make myprogram

myprogram is the target, it is built from one or more files, the prerequisites or dependents.

The dependencies are specified in a description file (Makefile or makefile), together with the commands to be executed to build the target.

Example:

```
program: main.o iodat.o dorun.o lo.o /usr/lib/crtn.a
    cc -o program main.o iodat.o dorun.o lo.o \
    /usr/lib/crtn.a

main.o : main.c
    cc -c main.c

iodat.o : iodat.c
    cc -c iodat.c

dorun.o : dorun.c
    cc -c dorun.c

lo.o : lo.s
    as -o lo.o lo.s
```

Note that there are dependency lines (or rule lines) with a ":" and command lines, starting with a tab character.

The example is rather clumsy, repetitive. Things become simpler by using macros and by exploiting the suffix rules.

A macro is defined as:

name = a text string

You refer to a macro as:

$(name) or ${name}

Example:

```
LIBES = -lX11
objs = drawable.o plot_points.o root_data.o
CC = /usr/bin/gcc
23 = "This is the 23rd run"
OPT =
DEBUG_FLAG =        #empty now, use later
BINDIR = /usr/local/bin
plot : $(objs)
    $(CC) -o plot $(DEBUG_FLAG) $(objs) $(LIBES)
    mv plot $(BINDIR)
```

When you now type: **make plot**
the shell will receive the following commands:

```
/usr/bin/gcc -o plot drawable.o plot_points.o \
root_data.o -lX11
mv plot /usr/local/bin
```

Macros can be nested. The order of definition is immaterial **make** also has macros defined internally.

Shell **environment variables** can be used as macros in the **Makefile**. So, if you have a shell environment variable

DIR = /usr/proj and you have *exported* it:
export DIR

then you may use **$(DIR)** in your Makefile:

```
SRC = $(DIR)/src
myprog :
    cd $(SRC) ; ....
```

As macros can be defined in many places, there are **priority rules**, to avoid conflicts.

Special internal macros:

$(@) (or **$@**) evaluates to the **current target**
$? evaluates to a **list of prerequisites, newer than the current target.**

The other important concept is the **suffix rules**:

```
.SUFFIXES : .o .c .s
.c.o :
    $(CC) $(CFLAGS) -o $@ $<
.s.o :
    $(AS) $(ASFLAGS) -o $@ $<
```

$<** is the same as **$?, but for use in suffix rules only.

Our first example now becomes:

```
OBJS = main.o lodat.o dorun.o lo.o
LIB = /usr/lib/crtn.a

program : $(OBJS) $(LIB)
    $(CC) -o $@ $(OBJS) $(LIB)
```

These are the very essentials of **make**. For many more details, see the **man pages** or:

A. Oram and S. Talbott, *Managing Projects with make*, O'Reilly & Associates, 1991, Sebastopol CA, ISBN 0-937175-90-0

The examples were taken from this book.

make is absolutely **essential** for developing software of some complexity, especially if done by a team of programmers.

X11 - X11

X11 was developed at **MIT**. The version for the **Intel processors** is called **Xfree86**, and is installed on your machines.

When you log in, **X11 starts up automatically** and you will find at least **one window** which **emulates a terminal (Xterm)** and from which you can work.

You may open other windows, move them around, login to several Xterm windows, run a number of programs simultaneously in different windows (for instance you may have a window open for your editor), etc. etc. You may use **Xview**, which gives you a few more possibilities.

Play with all this and get accustomed to it.

Later you will also learn how to write programs which make use of the X11 facilities.

XIII - How and where do I get Linux?

There are various distributions of Linux: Debian, je, MCC, **Slackware** and half a dozen others.

Slackware is one of the most popular and most complete. It is frequently updated, which some people consider to be a disadvantage.

Slackware was installed on our machines.

You can obtain the **Linux distributions** via **ftp** or on **CD-ROM** and sometimes you can install directly over Internet.

If you cannot ftp, then a friend may be able to produce ~60 **1.44 M floppies** for you, for the complete system, including many things you may not need or want.

To know more about the various distributions or where to obtain CD-ROMs, first consult **the list of anonymous ftp sites** which distribute Linux. When you connect to an anonymous ftp site, you may need to do some searching through directories.

By way of example, I give the directories on
nic.switch.ch

You should really try to connect to the **ftp site nearest to your home**. See the list of ftp sites which distribute Slackware.

This list was taken from

/mirror/linux/distributions/slackware
on **nic.switch.ch**

The other distributions can be found in subdirectories of
/mirror/linux/distributions

There you also find a file **Distribution-HOWTO** which gives detailed information on the various distributions, with a bias toward Slackware.

If you want to buy a CD-ROM, you should also be able to find addresses, etc. Probably in a directory **...../distributions/cdrom**.

Supposing you have decided for **Slackware**, then the next thing is to get the **installation-HOWTO** from **/mirror/linux/distributions/slackware**.

Read it first before you ftp further.

The **slackware** distribution comes **in chunks** which correspond to a **1.44 Mbyte diskette**. (5¼" 1.2 Mbyte diskettes can also be had, with some trouble during installation).

Some "**series**" of diskettes **are absolutely needed** to run Linux, others are **necessary** if you want to **develop software** to run under Linux. Still others are "**recommended**" or "**optional**" or even pure luxury, but nice to have if your disk can hold them.

The **installation-HOWTO** gives details and also tells you **how large your machine and disk** should be in various cases.

Note that you need a **386** or **486**.

If you have to install from floppies, write out your a1, a2, etc. directories, each one to a floppy (or ftp directly to the floppies).

When you have made up your mind which series you want, go to the subdirectories **a1, a2, a3, ap1,** ap2, etc. etc. of **...../slackware** one after the other and do a **binary** ftp: **mget ***

Do not forget to set up the corresponding series of **directories a1, a2, a3,** etc. **on your receiving machine** and to receive each "diskette image" into the correct directory.

Apart from a1, a2, etc. there is a subdirectory **install**. Get from here the **boot and root disk images** of your choice. Your choice will depend on the **type of disk controller** you have and if you are connected to a **network** or not.

Also get RAWRITE.EXE and UNZIP.EXE from the **install** directory.

XIV - How do I install Linux?

Follow precisely the steps described in the Installation-HOWTO.

Installation from floppies will take 1½ hours for the complete system. The time will depend on your a priori knowledge.

In the event that you have space to waste on your disk, you may ftp everything to **your DOS partition and install Linux from there.** Installation time will be reduced to approximately 20 minutes.

Before you can start installing Linux (from floppies, from DOS or from CD-ROM) you must make your "boot" and "root" disks using RAWRITE.EXE and you must **set up a partition for Linux and a swap partition.** Use these **boot and root disks** also if you want to add other "series" to your existing system.

If you have a very modern machine **with 2.88 Mbyte floppy** drive(s), you must set up your machine for "2.88Mbyte as 1.44 Mbyte".

If you do not do this, Linux will be unable to produce a boot floppy. Without a boot floppy you have no safety net!

XV - How do I get more information?

The **i and f series** of the Slackware distribution are full of extra information: **Guides and HOWTOs for the i series, FAQs for f.**

If you cannot install them on your machine, because of limited disk space, it will be worthwhile getting printed versions.

Some can be printed straight away (ASCII files), others may need a LaTex treatment and dvips (the Guides, in particular).

If these do not satisfy your appetite, then subscribe to one of the **Linux newsgroups**, for instance

comp.os.linux.

There are other Linux-related newsgroups, accessible from most UNIX installations.

If you are still unsatisfied you may take out a subscription to the monthly "**Linux Journal**", published by **Specialized Systems Consultants, Inc. (SSC).**

Subscription orders for Linux Journal to:
Linux Journal,
P.O. Box 85867,
Seattle, WA 98145-1867.
e-mail: subs@ssc.com
phone: (206)527-3385
fax: (206)527-2806.

O'Reilly Associates specialize in publishing books describing **GNU** and **X11** software packages.

"**The Linux Bible**" can be bought from: **Yggdrasil,**
phone: (800)261-6630, (408)261-6630,
fax: (408)261-6631.

This information on "where to get more information" is probably very incomplete.