Rute Users Tutorial and Exposition

http://www.obsidian.co.za/rute/

Paul Sheer

March 28, 2000

Copying

This license dictates the conditions under which you may copy, modify and distribute this work.

This license may be updated at a later date to better reflect the ideas of the "Open Content" movement. At the moment, the Open Content community is still debating a license. When their license is complete, this work will most probably adopt it. See http://www.opencontent.org/ for more details.

Exceptions to this license are likely to be granted for specific cases. Please email the author for details.

TERMS AND CONDITIONS

1. This work may not be reproduced in hard copy except for personal use. Further, it may not be reproduced in hard copy for training material, nor for commercial gain, nor for public or organisation-wide distribution. Further, it may not be reproduced in hard copy except where the intended reader of the hard copy initiates the process of converting the work to hard copy.

2. The work may not be modified except by a generic format translation utility, as may be appropriate for viewing the work using an alternative electronic media. Such a modified version of the work must clearly credit the author, display this license, and include all copyright notices. Such a modified version of the work must clearly state the means by which it was translated, as well as where an original copy can be obtained.

3. Verbatim copies of the work may be redistributed through any electronic media. Modified versions of the work as per 2. above may be redistributed same, provided that they can reasonably be said to include, albeit in translated form, all the original source files.

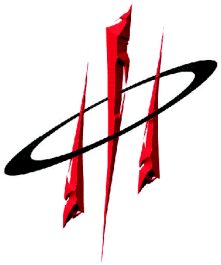
4. The work is not distributed to promote any product, computer program or operating system. Even if otherwise cited, all of the opinions expressed in the work are exclusively those of the author. The author withdraws any implication that any statement made within this work can be justified, verified or corroborated.

NO WARRANTY

5. THE COPYRIGHT HOLDER(S) PROVIDE NO WARRANTY FOR THE ACCURACY OR COMPLETENESS OF THIS WORK, OR TO THE FUNCTIONALITY OF THE EXAMPLE PROGRAMS OR DATA CONTAINED THEREIN, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

6. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE WORK AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE WORK (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

info@obsidian.co.za
+27 11 792-6500
+27 21 448-9265
Linux
Installations — Support — Networking
Intranet services — Web hosting — Firewalls
Development — Training — Outsourcing



Obsidian Systems (Pty) Ltd

Contents

Chapter 1

Introduction

1.1 Read this first

Rute must be read from beginning to end, in consecutive order. You must also practice each example.

1.2 About Rute

Chapter 15 contains a fairly comprehensive list of all reference documentation available on your system. Rute aims to supplement this material with a tutorial that is both comprehensive and independent of any previous UNIX knowledge.

Rute is a *dependency consistent* tutorial document. This means that you can (and must) read it from beginning to end in consecutive order.

Rute also satisfies the requirements for course notes for a LINUX training course. Here in South Africa, the initial part of Rute is being used for a 16 hour LINUX training course given in eight lessons.

1.3 Is this stuff for beginners? How can I learn *advanced* administration?

On UNIX, anyone who can read system documentation can set up any service. No one need *teach* you how to set up an Internet service like Web, mail, FTP, SQL etc. and there are no books worth buying on these subjects.

If you don't know where to find the documentation for these packages on your file system, or they don't make sense to you when you read them, then you need to read Rute (again).

Creativity and ingenuity are requirements for being a UNIX administrator. They are your own responsibility. Never before has everything you ever needed to do anything, been available in a single package — this is LINUX. How do I _____? There is one answer: it is already on your computer, waiting for you to read it.

1.4 I get very frustrated with Unix documentation that I don't understand

Any system reference will require you to read it at least three times before you get a reasonable picture of what to do. If you need to read it more than three times, then there is probably some other information that you really should be reading first. If you are only reading a document once, then you are being too impatient with yourself.

It is very important to identify the exact terms that you fail to understand in a document. Always try to back-trace to the precise word before you continue.

It is usually cheaper and faster to read a document three times than to pay someone to train you. Don't be lazy.

Don't learn new things according to deadlines. Your UNIX knowledge is going to evolve by grace and fascination, not by pressure.

Chapter 2

The GNU General Public License Version 2

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy  <name of author>

This program is free software; you can redistribute it and/or modify
```

```
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘`show w`’ and ‘`show c`’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘`show w`’ and ‘`show c`’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a ”copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.
>
> <*signature of Ty Coon*>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Chapter 3

Computing Sub-basics

The best way of thinking about how a computer stores and manipulates[1] information[2] is to ask yourself how *you* would. Most often the way a computer works is exactly the way one would expect it to if you were inventing it for the first time. The only limitation on this are those imposed by logical feasibility and imagination, but most anything else is allowed.

3.1 Files

Common to every computer system invented is the *file*. A file holds a single contiguous block of data. Any kind of data can be stored in a file and there is no data that cannot be stored in a file. Furthermore, there is no kind of data that is stored anywhere else except in a file. A file holds data of the same type, for instance a single picture will be stored in one file. A piece of text representing an essay will be stored in one file. It is uncommon for different types of data (say text and pictures) to be stored together in the same file because it is inconvenient. A computer will typical contain about 10000 files that have a great many purposes. Each file has its own name. The file name on a LINUX or UNIX machine can be up to 256 characters long and may contain letters, numbers and even some punctuation. The file name is usually explanatory — you might call a letter you wrote to your friend something like `Mary_Jones.letter` (from now on, whenever you see the typewriter font[3], it means that those are words that might be read of the screen of the computer). The name you choose has no meaning to the computer, and could just as well be any other combination of letters or digits, however you will refer to that data with that file name whenever you give an instruction to the computer regarding that data. It is important to internalize[4] the fact that computers do not have an interpretation for anything. A computer operates with a set of interdepenent[5] logical rules — it has no fixed way of working; for example, the reason a computer has files at all is because computer programmers have decided that that is the most universal and convenient way of storing data, and if you think about it, it really is.

The data in each file is merely a long list of numbers. The *size* of the file is just the length of the list of numbers. Each number is called a *byte* and can be from 0 to 255. This means that no file can contain any number over 255 or less than zero. Also, there is no type of data which cannot be represented as a list of numbers. Your letter to Mary will be *encoded* into numbers in order to be stored on the computer. We all know that a television picture is just a sequence of dots on the screen that scan from left to right — it is in this way that a picture might be represented in a file: i.e. as a sequence of numbers where each number is interpreted as a level of brightness; 0 for black, and 255 for white. For your letter, the convention is to store an `A` as 65 a `B` as 66 and so on. Each punctuation character also has a numerical equivalent.

[1] Manage skillfully
[2] Anything that you would like not to have to remember yourself
[3] A style of print.
[4] Think with that point of view.
[5] Without an apex.

3.2 Commands

The second thing common to every computer system invented is the *command*. You tell the computer what to do with single words — one at a time — typed into the computer. Modern computers appear to have done away the typing in of commands by having beautiful graphical displays that work with a mouse, but, fundamentally, all that is happening is that commands are being secretely typed in for you. Using commands is still the only way to have complete power over the computer. You don't really know anything about a computer until you get to grips with the commands it uses. Using a computer will very much involve typing in a word, pressing the enter key, and then waiting for the computer screen to spit something back at you. Most commands are typed in to do something useful to a file.

3.3 Logging in and changing your password

Turn on your LINUX box. You will then see the *login prompt*[6]. It will state the name of the computer (each computer has a name - typically consisting of about eight lowercase letters) and then word `login:`. Now you should type your *login name*[7], and then press the 'Enter' or 'Return' key[8]. A password prompt will appear afterwhich you should type your *password*[9], and then press the 'Enter' or 'Return' key again. The screen might show some message and prompt you for a login again - in this case you have probably typed something incorrectly and should give it another try. From now on, you will be expected to know that the 'Enter' or 'Return' key should be pressed at the end of every line you type in, analogous to the mechanical typewriter. You will also be expected to know that human error is very common — when you type something incorrectly the computer will give an error message, and you should try again until you get it right. It is very uncommon for a person to understand computer concepts after a first reading, or to get commands to work the first try.

Now that you have logged in, you will see a shell[10]prompt. This is where you will spend most of your time as a system adminstrator[11], but it needn't look as bland you see now. Your first excersize is to change your password. Type the command `passwd`. You will then be asked for a new password and then asked to confirm that password[12]. Then you will arrive back in the shell. The password you have chosen will take effect immediately, replacing the previous password that you used to log in. The password command might also have given some message indicating what effect it actually had. You may not understand the message, but you should try to get an idea of whether the conatation[13] was positive or negative.

When you are using a computer, it is useful to imagine yourself as *being* in different places *within* the computer, rather than just typing commands into it. After you entered the `passwd` command, you were no longer *in* the shell, but moved *into* the password *place*. You could not use the shell until you had move out of the `passwd` command.

3.4 Listing and creating files

Type in the command `ls`. `ls` is short for *list*, abbreviated to two letters like most other UNIX commands. `ls` will list all your current files. You may find that `ls` does nothing, but just returns you back to the shell. This would be because you have no files just yet. Most UNIX commands do *not* give any kind of message unless something went wrong (the `passwd` command above was an exception). If there were files, you would see their names listed rather blandly in columns with no indication of what they are for.

There are many ways of creating a file. Type `cat > Mary_Jones.letter` and then type out a few lines of text. You will use this file in later examples. The `cat` command is used here to write from the keyboard into a file

[6] A prompt is one or more characters displayed on the screen that you are expected to type something in after.
[7] A sequence of about eight lower case letters that would have been assigned to you by your computer administrator or some other source.
[8] Some keyboards have an key labelled 'Enter' while others label the same key 'Return'.
[9] Your password *may* be the same as your *login name*. Note that your password will not be shown on the screen as you type it, but will be invisible.
[10] A place where you are able to type commands.
[11] Computer manager.
[12] The password you choose consist of letters, numbers and punctuation — you will see later on why this security measure is a good idea. Take good note of your password for the next time you log in.
[13] Implied meaning.

`Mary_Jones.letter`. At the end of the last line, press Enter one more time and then press Ctrl-C[14]. Now, if you type `ls` again, you will see the file `Mary_Jones.letter` listed with any other files. Type `cat Mary_Jones.letter` **without** the `>`. You will see that the command `cat` writes the contents of a file to the screen, allowing you to view your letter. It should match exactly what you typed in.

3.5 Directories

Before we mentioned that a system may typically contain 10000 files. It would be combersome if you were to see all 10000 of them whenever you typed `ls`, hence files are placed in different "cabinets" so that files of the same type get placed together and can be easily isolated from other files. For instance your letter above might go in a seperate "cabinet" with other letters. A "cabinet" in computer terms is actually called a directory. This is the third commonality between all computer systems: all files go in one or other directory. To get and idea of how this works, type the command `mkdir letters` where `mkdir` stands for *make directory*. Now type `ls`. This will show the file `Mary_Jones.letter` as well as a new file `letters`. The file `letters` is not really a file at all, but the name of a directory in which a number of other files can be placed. To go *into* the directory `letters` you can type `cd letters` where `cd` stands for *change directory*. Since the directory is newly created, you would not expect it to contain any files, and typing `ls` will verify this by not listing anything. You can now create a file using the `cat` command as you did before (try this). To go back into the original directory that you were in you can use the command `cd ..` where the `..` has a special meaning of taking you out of the current directory. Type `ls` again to verify that you have actually gone *up* a directory.

It is however bothersome that one cannot tell the difference between files and directories. The way to do this is with the `ls -l` command. `-l` stands for *long* format. If you enter this command you will see a lot of details about the files that may not yet be comprehensible to you. The three things you can watch for are the filename on the far right, the file size (i.e. the number of bytes that the file contains) in the third column from the right, and the file type on the far left. The file type is a string of letters of which you will only be interested in one: the character on the far left is either a `-` or a `d`. A `-` indicates a regular file, and a `d` indicates a directory. The command `ls -l Mary_Jones.letter` will list only the single file `Mary_Jones.letter` and is useful for finding out the size of a single file.

In fact, there is no limitation on how many directories you can create within each other. In what follows, you will get a glimpse of the layout of all the directories on the computer.

Type the command `cd /` where the `/` has the special meaning to go to the top most directory on the computer called the *root* directory. Now type `ls -l`. The listing may be quite long and may go off the top of the screen. You will see that most, if not all, are directories. You can now practice moving around the system with the `cd` command, not forgetting that `cd ..` takes you up and `cd /` takes you to the root directory.

When you have finished, log out of the computer using the `logout` command.

[14] The **Ctrl** key held down while pressing the **C** key.

Chapter 4

Basic Commands

> *All of* UNIX *is* case sensitive. *A command with even a single letter's capitalisation altered, is considered to be a completely different command. The same goes for files, directories, configuration file formats and the syntax all native programming languages.*

4.1 The `ls` command, hidden files, command-line options

In addition to directories and ordinary text files, there are other types of files, although all files contain the same kind of data (i.e. a list of bytes). The *hidden* file is a file that will not ordinarily appear when you type the command `ls` to *list* the contents of a directory. To see a hidden file you have to use the command `ls -a`. The `-a` option means to list *all* files as well as hidden files. Another varient `ls -l` which lists the contents in *long* format. The `-` is used in this way to indicate variations on a command. These are called *command-line options* or *command-line arguments*, and most UNIX commands can take a number of them. They can be strung together in any way that is convenient[1], for example `ls -a -l`, `ls -l -a` or `ls -al` — either of these will list *all* files in *long* format.

All GNU commands take an additional argument `-h` and `--help`. You can type a command with just this on the command line and get a *usage summary*. This is some brief help that will summarise options that you may have forgotten if you are **already** familiar with the command — it will never be an exhaustive description of the usage. See the later explanation about `man` pages.

The difference between a *hidden* file and an ordinary file is merely that the file name of a *hidden* file starts with a period. Hiding files in this way is not for security, but for convenience.

The option `ls -l` is somewhat cryptic for the novice. Its more explanatory version is `ls --format=long`. Simlarly, the *all* option can be given as `ls --all`, and will mean the same thing as `ls -a`.

4.2 Error messages

Although commands usually do not display a message when they *execute*[2] succesfully, commands do report *errors* in a consistant format. The format will vary from one command to another, but will often appear as follows: *command-name: what was attempted: error message*. For example, the command `ls -l qwerty` will give an error `ls: qwerty: No such file or directory`. What actually happened was that the command `ls` attempted to read the file `qwerty`. Since this file does not exist, an error code 2 arose. This error code corresponds to a situation when a file or directory is not found. The error code is automatically translated into the sentance `No such file or directory`. It is important to understand the distinction between an explanatory message that a command gives (such as the messages reported by the `passwd` command in the previous chapter) and an

[1] Commands under the GNU free software license are superior in this way: they have a greater number of options than traditional UNIX commands and are therefore more flexable.

[2] The computer accepted and processed the command.

error code that was just translated into a sentance. This is because a lot of different kinds of problems can result in an identical error code (there are only about a hundred different error codes). Experience will teach you that error messages do **not** tell you what to do, only what went wrong, and should not be taken as gospel.

A complete list of basic error codes can be found in /usr/include/asm/errno.h. In addition to these, several other header file[3] may define there own error code. Under UNIX however, these are 99% of all the errors you are ever likely to get. Most of them will be meaningless to you at the moment, but are included here as a reference:

```
#ifndef _I386_ERRNO_H
#define _I386_ERRNO_H

#define EPERM            1      /* Operation not permitted */
#define ENOENT           2      /* No such file or directory */
#define ESRCH            3      /* No such process */
#define EINTR            4      /* Interrupted system call */
#define EIO              5      /* I/O error */
#define ENXIO            6      /* No such device or address */
#define E2BIG            7      /* Arg list too long */
#define ENOEXEC          8      /* Exec format error */
#define EBADF            9      /* Bad file number */
#define ECHILD          10      /* No child processes */
#define EAGAIN          11      /* Try again */
#define ENOMEM          12      /* Out of memory */
#define EACCES          13      /* Permission denied */
#define EFAULT          14      /* Bad address */
#define ENOTBLK         15      /* Block device required */
#define EBUSY           16      /* Device or resource busy */
#define EEXIST          17      /* File exists */
#define EXDEV           18      /* Cross-device link */
#define ENODEV          19      /* No such device */
#define ENOTDIR         20      /* Not a directory */
#define EISDIR          21      /* Is a directory */
#define EINVAL          22      /* Invalid argument */
#define ENFILE          23      /* File table overflow */
#define EMFILE          24      /* Too many open files */
#define ENOTTY          25      /* Not a typewriter */
#define ETXTBSY         26      /* Text file busy */
#define EFBIG           27      /* File too large */
#define ENOSPC          28      /* No space left on device */
#define ESPIPE          29      /* Illegal seek */
#define EROFS           30      /* Read-only file system */
#define EMLINK          31      /* Too many links */
#define EPIPE           32      /* Broken pipe */
#define EDOM            33      /* Math argument out of domain of func */
#define ERANGE          34      /* Math result not representable */
#define EDEADLK         35      /* Resource deadlock would occur */
#define ENAMETOOLONG    36      /* File name too long */
#define ENOLCK          37      /* No record locks available */
#define ENOSYS          38      /* Function not implemented */
#define ENOTEMPTY       39      /* Directory not empty */
#define ELOOP           40      /* Too many symbolic links
                                                    encountered */
#define EWOULDBLOCK     EAGAIN  /* Operation would block */
#define ENOMSG          42      /* No message of desired type */
#define EIDRM           43      /* Identifier removed */
#define ECHRNG          44      /* Channel number out of range */
#define EL2NSYNC        45      /* Level 2 not synchronized */
#define EL3HLT          46      /* Level 3 halted */
#define EL3RST          47      /* Level 3 reset */
#define ELNRNG          48      /* Link number out of range */
#define EUNATCH         49      /* Protocol driver not attached */
#define ENOCSI          50      /* No CSI structure available */
```

[3] Files ending in .h

```
#define EL2HLT          51      /* Level 2 halted */
#define EBADE           52      /* Invalid exchange */
#define EBADR           53      /* Invalid request descriptor */
#define EXFULL          54      /* Exchange full */
#define ENOANO          55      /* No anode */
#define EBADRQC         56      /* Invalid request code */
#define EBADSLT         57      /* Invalid slot */

#define EDEADLOCK       EDEADLK

#define EBFONT          59      /* Bad font file format */
#define ENOSTR          60      /* Device not a stream */
#define ENODATA         61      /* No data available */
#define ETIME           62      /* Timer expired */
#define ENOSR           63      /* Out of streams resources */
#define ENONET          64      /* Machine is not on the network */
#define ENOPKG          65      /* Package not installed */
#define EREMOTE         66      /* Object is remote */
#define ENOLINK         67      /* Link has been severed */
#define EADV            68      /* Advertise error */
#define ESRMNT          69      /* Srmount error */
#define ECOMM           70      /* Communication error on send */
#define EPROTO          71      /* Protocol error */
#define EMULTIHOP       72      /* Multihop attempted */
#define EDOTDOT         73      /* RFS specific error */
#define EBADMSG         74      /* Not a data message */
#define EOVERFLOW       75      /* Value too large for defined data
                                                        type */
#define ENOTUNIQ        76      /* Name not unique on network */
#define EBADFD          77      /* File descriptor in bad state */
#define EREMCHG         78      /* Remote address changed */
#define ELIBACC         79      /* Can not access a needed shared
                                                         library */
#define ELIBBAD         80      /* Accessing a corrupted shared
                                                         library */
#define ELIBSCN         81      /* .lib section in a.out corrupted */
#define ELIBMAX         82      /* Attempting to link in too many shared
                                                       libraries */
#define ELIBEXEC        83      /* Cannot exec a shared library
                                                         directly */
#define EILSEQ          84      /* Illegal byte sequence */
#define ERESTART        85      /* Interrupted system call should be
                                                       restarted */
#define ESTRPIPE        86      /* Streams pipe error */
#define EUSERS          87      /* Too many users */
#define ENOTSOCK        88      /* Socket operation on non-socket */
#define EDESTADDRREQ    89      /* Destination address required */
#define EMSGSIZE        90      /* Message too long */
#define EPROTOTYPE      91      /* Protocol wrong type for socket */
#define ENOPROTOOPT     92      /* Protocol not available */
#define EPROTONOSUPPORT 93      /* Protocol not supported */
#define ESOCKTNOSUPPORT 94      /* Socket type not supported */
#define EOPNOTSUPP      95      /* Operation not supported on transport
                                                         endpoint */
#define EPFNOSUPPORT    96      /* Protocol family not supported */
#define EAFNOSUPPORT    97      /* Address family not supported by
                                                         protocol */
#define EADDRINUSE      98      /* Address already in use */
#define EADDRNOTAVAIL   99      /* Cannot assign requested address */
#define ENETDOWN        100     /* Network is down */
#define ENETUNREACH     101     /* Network is unreachable */
#define ENETRESET       102     /* Network dropped connection because
                                                         of reset */
#define ECONNABORTED    103     /* Software caused connection abort */
```

```
#define ECONNRESET      104     /* Connection reset by peer */
#define ENOBUFS         105     /* No buffer space available */
#define EISCONN         106     /* Transport endpoint is already
                                                        connected */
#define ENOTCONN        107     /* Transport endpoint is not connected */
#define ESHUTDOWN       108     /* Cannot send after transport endpoint
                                                         shutdown */
#define ETOOMANYREFS    109     /* Too many references: cannot splice */
#define ETIMEDOUT       110     /* Connection timed out */
#define ECONNREFUSED    111     /* Connection refused */
#define EHOSTDOWN       112     /* Host is down */
#define EHOSTUNREACH    113     /* No route to host */
#define EALREADY        114     /* Operation already in progress */
#define EINPROGRESS     115     /* Operation now in progress */
#define ESTALE          116     /* Stale NFS file handle */
#define EUCLEAN         117     /* Structure needs cleaning */
#define ENOTNAM         118     /* Not a XENIX named type file */
#define ENAVAIL         119     /* No XENIX semaphores available */
#define EISNAM          120     /* Is a named type file */
#define EREMOTEIO       121     /* Remote I/O error */
#define EDQUOT          122     /* Quota exceeded */

#define ENOMEDIUM       123     /* No medium found */
#define EMEDIUMTYPE     124     /* Wrong medium type */

#endif
```

4.3 Wildcards, names, extensions and *glob* expressions

`ls` can produce a lot of output if there are a large number of files in a directory. Now say that we are only interested in files that ended with the letters `tter`. To list only these files you can use `ls *tter`. The `*` matches any number of any other characters. So, for example, the files `Tina.letter`, `Mary_Jones.letter` and the file `splatter`, would all be listed if they were present. While a file `Harlette` would not be listed. While the `*` matches any length of characters, then `?` matches only one character. For example the command ls `?ar*` would list the files `Mary_Jones.letter` and `Harlette`.

When naming files, it is a good idea to choose names that group files of the same type together. We do this by adding an *extension* to the file name that describes the type of file it is. We have already demonstrated this by calling a file `Mary_Jones.letter` instead of just `Mary_Jones`. If you keep this convention, you will be able to easily list all the files that are letters by entering `ls *.letter`. The file-name `Mary_Jones.letter` is then said to be composed of two parts: the *name*, `Mary_Jones` and the *extension*, `letter`.

Some extentions you may see are

`.a` Archive. `lib*.a` is a static library.

`.alias` X Window System font alias catalogue.

`.avi` Video format.

`.au` Audio format.

`.awk` `awk` program source file.

`.bib` `bibtex` LaTeXbibliography source file.

`.bmp` Microsoft Bitmap file image format.

`.bz2` File compressed with the `bzip2` compression program.

`.cc`, `.cxx`, `.C`, `.cpp` C++ program source code.

`.cf`, `.cfg` Configuration file or script.

`.cgi` Executable script that produces web page output.

`.conf`, `.confif` Configuration file.

`.csh` `csh` Shell script.

`.c` C program source code.

`.db` Database file.

`.dir` X Window System font/other database directory.

`.deb` Debian package for the Debian distribution.

`.diff` Output of the diff program indicating the difference between files or source trees.

`.dvi` Device independent file. Formatted output of `.tex` LaTeX file.

`.el` Lisp program source.

`.gif`, `.giff` Giff image file.

`.gz` File compressed with the `gzip` compression program.

`.htm`, `.html`, `.shtm`, `.html` Hyper Text Markup Language. A web page of some sort.

`.h` C/C++ program header file.

`.i` SWIG source, or **C** preprocessor output.

`.in` `configure` input file.

`.info` Info pages read with the `info` command.

`.jpg`, `.jpeg` JPEG image file.

`.lj` LaserJet file. Suitable input to a HP laserjet printer.

`.log` Log file of a system service. This file grows with status messages of some system program.

`.lsm` LINUX Software Map entry.

`.lyx` LyX word processor document.

`.man` Man page.

`.mf` Meta-Font font program source file.

`.pbm` PBM image file format.

`.pcf` PCF image file — intermediate representation for fonts. X Window System font.

`.pcx` PCX image file.

`.pfb` X Window System font file.

`.pdf` Formatted document similar to postscript or dvi.

`.php` PHP program source code (used for web page design).

`.pl` Perl program source code.

`.ps` PostScript file, for printing or viewing.

`.py` Python program source code.

`.rpm` RedHat Package Manager `rpm` file.

`.sgml` Standard Generalized Markup Language. Used to create documents to be converted to many different formats.

`.sh` `sh` Shell script.

`.so` Shared object file. `lib*.a` is a *Dynamically Linked Library*[4].

`.spd` Speedo X Window System font file.

`.tar` `tar`red directory tree.

`.tcl` Tcl/Tk source code (programming language).

`.texi`, `.texinfo` Texinfo source. This is from what info pages are compiled.

`.tex` TEX or LATEX document. LATEX is for document processing and typesetting.

`.tga` TARGA image file.

`.tgz` `tar`red and `gzip`ped directory tree. Also a for the package Slackware distribution.

`.tiff` Tiff image file.

`.tfm` LATEXfont metric file.

`.ttf` True type font.

`.txt` Plain English text file.

`.xpm` XPM image file.

`.y` `yacc` source file.

`.Z` File compressed with the `compress` compression program.

`.zip` File compressed with the `pkzip` (or `pkzip` DOS) compression program.

`.1`, `.2` `...` Man page.

In addition, files that have no extension and a capitalised descriptive name are usually plain English text and meant for your reading. This will come bundled with packages and are for documentation purposes.

Some full file names you may see are

`AUTHORS` List of people who contributed to or wrote a package.

`ChangeLog` List of developer changes made to a package.

`COPYING` Copyright (usually GPL) for a package.

`INSTALL` Instalation instructions.

`README` Help information to be read first, pertaining to the directory the `README` is in.

`TODO` List of future desired work to be done to package.

`BUGS` List of errata.

`NEWS` Info about new features and changes for the layman about this package.

`THANKS` List of contributors to a package.

`VERSION` Version information of the package.

There is also a way to restrict characters of a file-name within certain ranges, like if you only want to list the files that begin with A through M, you can do `ls [A-M]*`. Here the brackets have a special meaning — they match a single character like a `?`, but only those given by the range. You can use this in a variety of ways, for example `[a-dJW-Y]*` matches all files beginning with `a`, `b`, `c`, `d`, `J`, `W`, `X` or `Y`, `*[a-d]id` matches all files ending with `aid`, `bid`, `cid` or `did`. This way of specifying a file-name is called a *glob* expression. *Glob* expressions are used in many different contexts as you will see later.

[4] Executable program code shared by more than one program in the `bin` directory to save disk space and memory.

4.4 Usage summaries and the copy command

The command cp stands for *copy* and is used to make a duplicate of one file or a number of files. The format is

```
cp <file> <newfile>
cp <file> [<file> ...] <dir>
```

The above lines are called a *usage summary*. The < and > signs mean that you don't actually type out these characters but replace <file> with a file-name of your own. These are also sometime written in italics like, cp *file newfile*. <file> and <dir> are called a *parameters*. This is a common convention used to specify the usage of a command. The [and] brackets are also not actually typed but mean that the contents between them are optional. The ellipses ... mean that <file> can be be given repeatedly, and these also are never actually typed. From now on you will be expected to substitute your own parameters by interpretting the usage summary. You can see that the second of the above lines is actually just saying that one or more file names can be listed with a directory name last.

From the above usage summary it is obvious that there are two ways to use the cp command. If the the last name is not a directory then cp will copy that file and rename it to the filename given. If the last name is a directory then cp will copy all the files listed *into* that directory.

The usage summary of the ls command would be as follows:

```
ls [-l, --format=long] [-a, --all] <file> <file> ...
```

where the comma indicates that either option is valid. Similarly with the passwd command:

```
passwd [<username>]
```

You should practice using the cp command now by moving some of your files from place to place.

4.5 Manipulating directories

The cd command is used to take you to different directories. Create a directory new with mkdir new. You *could* create a directory one by doing cd new and then mkdir one, but there is a more direct way of doing this with mkdir new/one. You can then change directly to the one directory with cd new/one. And similarly you can get back to where you were with cd ../... In this way, the / is used to represent directories within directories. The directory one is called a *subdirectory* of new.

The command pwd stands for *present working directory* (also called the *current directory*) and is used to tell you what directory you are currently in. Entering pwd will give you some output like /home/<username>. Experiment by changing to the root directory (with cd /) and then back into the directory /home/<username> (with cd /home/<username>). The directory /home/<username> is called your *home directory*, and is where all your personal files are kept. It can be used at any time with the abbreviation ~. In other words, entering cd /home/<username> is the same as entering cd ~. The process whereby a ~ is substituted for your home directory is called *tilde expansion*.

To remove a file use the command rm <filename>. To remove a directory use the command rmdir <dir>. Practice using these two commands. Note that you cannot remove a directory unless it is empty. To remove a directory as well as any contents it might contain, use the command rm -R <dir>. The -R option indicates to dive into any subdirectories of <dir> and delete their contents. The process whereby a command dives into subdirectories of subdirectories etc.is called recursion. -R stands for *recursively*. This is a very dangerous command.

The cp command also takes the -R option, allowing it to copy whole directories. The mv command is used to move files and directories. It works just like cp, but deletes the source files.

4.6 *Relative* vs. *absolute* pathnames

A command that requires a file name can be given the file in two ways. If you are in the same directory as the file (i.e. the file is in the current directory), then you can just enter the file name on its own (eg. `cp my_file new_file`). Otherwise, you can enter the *full path name*, like `cp /home/jack/my_file /home/jack/new_file`. Very often administrators use the notation `./my_file` to be clear about the distinction, for instance: `cp ./my_file ./new_file`. The leading `./` makes it clear that both files are relative to the current directory.

4.7 System manual pages

(See Chapter 15 for a complete overview of all documentation on the system, and also how to print manual pages out in a properly typeset format.)

The command `man [<section>|-a] <command>` is used to get help on a particular topic and stands for *manual*. Every command on the entire system is documented in so named *man pages*[5]. Man pages are the authoritative reference on how a command works because they are usually written by the very programmer who created the command. Under UNIX, any printed documentation should be considered as being second hand information. Man pages however will often not contain the underlying concepts needed to understand in what context a command is used. Hence it is not possible for a person to learn about UNIX purely from man pages. However once you have the necessary background for a command, then its man page becomes an indispensable source of information and other introductory material may be discarded.

Now, man pages are divided into sections, numbered 1 through 9. Section 1 contains all man pages for system commands like the ones you have been using. Sections 2-7 also exist, but contain information for programmers and the like, which you will probably not have to refer to just yet. Section 8 contains pages specifically for system administration commands. There are some additional sections labelled with letters; other than these, there are no manual pages outside of the sections 1 through 9.

You should now use the `man` command to look up the manual pages for all the commands that you have learned. Type `man cp`, `man mv`, `man rm`, `man mkdir`, `man rmdir`, `man passwd`, `man cd`, `man pwd` and of course `man man`. Much of the information may be incomprehensible to you at this stage. Skim through the pages to get an idea of how they are structured, and what headings they usually contain.

4.8 System `info` pages

`info` pages contain some excellent reference and tutorial information in hypertext linked format. Type `info` on its own to go to the top level menu of the entire info hierarchy. You can also type `info <command>` for help on many basic commands. Some packages will however not have info pages.

4.9 Some basic commands

You should practice using each of these commands.

`bc` A calculator program that handles arbitrary precision (very large) numbers. It is useful for doing any kind of calculation on the command line. It use is left as an exercise.

`cal [[0-12] 1--9999]` Prints out a nicely formatted calender of the current month, or a specified month, or a specified whole year. Try `cal 1` for fun, and `cal 9 1752`, when the pope had a few days scrapped to compensate for round-off error in the Julian system.

`cat <filename> [<filename> ...]` Writes the contents of all the files listed to the screen. `cat` can join a lot of files together with `cat <filename> <filename> ... > <newfile>`. `<newfile>` will be an end on end *concatination* of all the files given.

[5] In the past few years a new format of documentation has evolved called *info*. These are considered the modern way to document commands, but most system documentation is still available only through `man`. There are very few packages that are not documented in man however.

`clear` Erases all the text in the current terminal.

`date` Prints out the current date and time. (The command `time` though does something entirely different.)

`du <directory>` Stands for *disk usage*, and prints out the amount of space occupied by a directory. It recurses into any subdirectories and can print only a summary with `du -s <directory>`. Also try du `--max-depth=1 /var`, and `du -x /`, on a system with `/usr` and `/home` on separate *partitions*[6].

`df` Stands for *disk free*. This tells you how much free space is left on your system. The available space usually has the units of kilobytes (1024 bytes) (although on some other UNIX systems this will be 512 bytes or 2048 bytes). The right most column tells the directory (in combination with any directories below that) under which that much space is available.

`dircmp` Directory compare. This can be used to compare directories to see if changes have been made between them. You will often want to see where two trees different (eg. check for missing files) possibly on different computers. Do a `man dircmp`. (This is a System 5 command and is not present on LINUX. You can however do directory comparisons with the Midnight Commander `mc`).

`free` Prints out available free memory. You will notice two listings: swap space and physical memory. These are contigous as far as the user is concerned. The swap space is a continuation of your installed memory that exists on disk. It is obviously slow to access, but provides the illusion of having much more RAM which avoids ever running out of memory (which can be quite fatal).

`echo` Prints a message to the terminal. Try `echo 'hello there'`, `echo $[10*3+2]`, `echo '$[10*3+2]'`. `echo -e` allows interpretation of certain *backslash* sequences, for example `echo -e "\a"`, which prints a *bell*, or in other words, beeps the terminal. `echo -n` does the same without printing the trailing newline. In other words it does not cause a wrap to the next line after the text is printed. `echo -e -n "\b"`, will print a back-space character only which will erase the last character printed.

`expr <expression>` Calculate the numerical expression `expression`. Most arithmetic operations that you are used to will work. Try `expr 5 + 10 '*' 2`. Observe how mathematical precidence is obeyed (i.e. the `*` is worked out before the `+`).

`file <filename>` This command prints out the type of data contained in a file. `file portrate.jpg` will tell you that `portrate.jpg` is a `JPEG image data, JFIF standard`. `file` detects an enormous amount of file types, accross every platform and computing standard.

`less` This is the GNU version of more, but has extra features. On your system the two commands may be the same. With `less`, you can use the arrow keys to page up and down through the file. You can do searches by pressing / then typing in a word to search for and then pressing Enter. Found words will be highlighted, and the text will be scrolled to the first found word.

`more` Displays a long file by stopping at the end of each page. Do the following: `ls -l /bin > bin.ls` then `more bin.ls`. The first command creates a file with the contents of the output of `ls`. This will be a long file because the directory `/bin` has a great many entries. The second command views the file. The space bar can be used to page through the file. When you get bored, just press q. you can also try `ls -l /bin | more` which will do the same thing in one go.

`sort <filename>` Prints out a file with lines sorted in alphabetical order. Create a file called `telephone` with each line containing a short telephone book entry. Then type sort `telephone`, or `sort telephone | less` and see what happens. Sort takes many interesting options to sort in reverse (`sort -r`), to eliminate duplicate entries (`sort -u`), ignore leading whitespace (`sort -b`), and so on. See the man page for details.

`strings <filename>` Prints out a binary file, but strips any unreadable characters. Readable groups of characters are placed on separate lines. If you have a binary file that you think may contain something interesting, but looks completely garbelled when viewed normally, use strings to sift out the interesting stuff: try `less /bin/cp` and then try `strings /bin/cp`.

`split ...` Splits a file into many seperate files. This might have been used when a file was too big to be copied onto a floppy disk and needed to be split into, say, 360kB pieces. Its sister, `csplit`, can split files along specified lines of text within the file. The commands are seldom used but are actually very useful when writing programs that manipulate text.

[6] See footnote on page 86

`uname` Prints out the name of the UNIX *operating system*[7] you are currently using.

`uniq <filename>` Prints out a file with duplicate lines deleted. The file must first be sorted.

`wc [-c] [-w] [-l] <filename>` Counts the number characters/bytes (with `-c`), words (with `-w`) or lines (with `-l`) in a file.

`whoami` Prints out your login name.

4.10 Compressed files

Files typically contain a lot of data they one can imagine might be represented with a smaller number of bytes. Take for example the letter you typed out. The word 'the' was probably repeated many times. You were probably also using lowercase letters most of the time. The file was by far not a completely random set of bytes and repeatedly used spaces as well as using some letters more than others[8]. Because of this the file can be *compressed* to take up less space. Compression envolves representing the same data using a smaller number of bytes, in such a way that the original data can be reconstructed exactly. This usually involves finding patterns in the data. The command to compress a file is `gzip <filename>` which stands for *GNU zip*. `gzip` a file in your home directory and then do an `ls` to see what happened. Now use `more` to view the compressed file. To uncompress the file you can use `gzip -d <filename>`. Now use `more` to view the file again. There are many files on the system which are stored in compressed format. For example man pages are often stored compressed and are uncompressed automatically when you read them.

You used the command `cat` previously to view a file. You can use the command `zcat` to do the same thing with a compressed file. Gzip a file and then type `zcat <filename>`. You will see that the contents of the file are written to the screen. Generally, when commands and files have a `z` in them they have something to do with compression — `z` stands for *zip*. You can use `zcat <filename> | less`, to view a compressed file proper. You can also use the command `zless <filename>`, which does the same as `zcat <filename> | less`. (Note that your `less` may actually have the functionality of `zless` combined.)

A new addition to the arsenal is `bzip2`. This is a compression program very much like `gzip`, except that it is slower and compresses 20%-30% better. It is useful for compressing files that will be downloaded from the Internet to reduce the transfer. Files that are compressed with `bzip2` have an extension `.bz2`. Note that the improvement in compression depends very much on the type of data being compressed. Sometimes there will be negligable improvement at the expense of a huge speed penalty, while occasionality there is a substantial reduction in size over `gzip` making it well worth it. Files that are frequently compressed and un-compressed should never use `bzip2`.

4.11 Searching for files

The command `find` can be used to search for files. Change to the root directory, and enter `find`. It will spew out all the files it can see by *recursively descending*[9] into all subdirectories. In other words, `find`, when executed while in the root directory, will print out all the files on the system. `find` will work for a long time if you enter it as you have — press Ctrl-C to stop it.

Now change back to your home directory and type `find` again. You will see *all* your personal files. There are a number of options `find` can take to look for specific files.

`find -type d` will show only directories and not the files they contain.

`find -type f` will show only files and not the directories that contain them, even though it will still descend into all directories.

`find -name <filename>` will find only files that have the name `<filename>`. For instance, `find -name '*.c'`. Will find all files that end in a `.c` extension (`find -name *.c` without the quote characters will not

[7] The *brand* of UNIX — there are number of different vendors and for each hardware platform.
[8] English text in fact contains, on average, only about 1.3 useful bits of data per byte.
[9] Goes into each subdirectory and all its subdirectories, and repeats the command `find`.

work. You will see why later). `find -name Mary_Jones.letter` will find the file with the name `Mary_Jones.letter`.

`find -size [[+|-]]<size>` will find only files that have a size larger (for `+`) or smaller (for `-`) than `<size>` kilobytes, or the same as `<size>` kilobytes if the sign is not specified.

`find <directory> [<directory> ...]` will start `find` in each of the directories given.

There are many more of these options for doing just about any type of search for a file. See the `find` man page for more details. `find` however has the deficiency of actively reading directories to find files. This is slow, especially when you start from the root directory. An alternative command is `locate <filename>`. This searches through a previously created database of all the files on the system, and hence finds files instantaneously. Its counterpart `updatedb` is used to update the database of files used by `locate`. On some systems `updatedb` runs automatically every day at 04h00.

4.12 Searching within files

Very often one would like to search through a number of files to find a particular word or phrase. An example might be where a number of files contain lists of telephone numbers with peoples names and addresses. The command `grep` does a line by line search through a file and prints out only those lines that contain a word that you have specified. grep has the command symmary,

```
grep [options] <pattern> <filename> [<filename> ...]
```

10

Do a grep for the word "the" to display all lines containing it: `grep 'the' Mary_Jones.letter`. Now try `grep 'the' *.letter`.

`grep -n <pattern> <filename>` will show the line number in the file where the word was found.

`grep -<num> <pattern> <filename>` will print out `<num>` of the lines that came before and after each of the lines in which the word was found.

`grep -A <num> <pattern> <filename>` will print out `<num>` of the lines that came `A`fter each of the lines in which the word was found.

`grep -B <num> <pattern> <filename>` will print out `<num>` of the lines that came `B`efore each of the lines in which the word was found.

`grep -v <pattern> <filename>` will print out only those lines that do *not* contain the word you are searching for[11].

`grep -i <pattern> <filename>` does the same as an ordinary `grep` but is case insensitive.

4.13 Copying to MSDOS and Windows formatted floppy disks

There is a package called the *mtools* package that enables reading and writing to MSDOS/Windows floppy disks. These are not standard UNIX commands but are packaged with most LINUX distributions. It supports long filename type floppy disks. Put an MSDOS disk in your `A:` drive. Try

[10] The words *word*, *string* or *pattern* are used synonomously in this context, basically meaning a short length of letters and/or numbers that you are trying to find matches for. A *pattern* can also be a string with kinds of wildcards in it that match different characters, as we shall see later.

[11] You may think that the `-v` option is no longer doing the same kind of thing that `grep` is advertised to do: i.e. *searching* for strings. In fact UNIX commands often suffer from this — they have such versatility that their functionality often overlaps with those of other commands. One actually never stops learning new and nifty ways of doing things hidden in the dark corners of man pages.

```
mdir A:
touch myfile
mcopy myfile A:
mdir A:
```

Note that there is **no** such thing as an `A:` disk under LINUX. Only the mtools package understand `A:` in order to retain familiarity for MSDOS users. The complete list of commands is

```
mattrib         mdeltree        mlabel          mrd
mbadblocks      mdir            minfo           mren
mcd             mdu             mmd             mshowfat
mcopy           mformat         mmount          mtoolstest
mdel            mkmanifest      mmove           mzip
                                mpartition      xcopy
```

and can be gotten by typing `info mtools`. In general you can take any MSDOS command, put it into lower case and add an `m` in front of it, to give you a command that you can use on LINUX.

4.14 Archives and backups

One of the primary activities of a system administrator is to make backups. It is a essential never to underestimate the volatility[12] of information in a computer. *Backups* are therefore continually made of data. A backup is a duplicate of your files, that can be used as a replacement should any or all of the computer be destroyed. The idea is that all of the data in a directory[13] are stored in a seperate place — often compressed — and can be retrieved in case of an emergency. When we want to store a number of files in this way, it is useful to be able to pack many files into one file so that we can perform operations on that single file only. When many files are packed together into one, this packed file is called an *archive*. File names of archives usually have an 'a' in their name somewhere. The most common being the extension `.tar`, which stands for *tape archive*.

To *create* an archive of a directory the `tar` command is used:

```
tar -c -f <filename> <directory>
```

Create a directory with a few files in it and run the `tar` command to back it up. A file of `<filename>` will be created. Take careful note of any error messages that `tar` reports. List the file and check that its size is appropriate for the size of the directory you are archiving. You can also use the *verify* option (see the man page) of the `tar` command to check the integrity of `<filename>`. Now remove the directory, and then restore it with the *extract* option of the `tar` command:

```
tar -x -f <filename>
```

You should see your directory recreated with all its files intact. A nice option to give to tar is `-v`. This will list all the files that are being added to or extracted from the archive as they are processed, and is useful to watch the progress of archiving. It is obvious that you can call your archive anything you like, however the common practice is to call it `<directory>.tar`, which makes it clear to all exactly what it is.

Once you have your tar file, you would probably want to compress it with `gzip`. This will create a file `<directory>.tar.gz`, which is sometimes also called `<directory>.tgz` for brevity.

A second kind of archiving utility is `cpio`. `cpio` is actually more powerful than tar, but is considered to be more cryptic to use. The principles of `cpio` are quite similar and its usage is left as an exercise.

4.15 The `PATH` where commands are searched for

When you type a command at the shell prompt, it has to be read off disk out of one or other directory. On UNIX all such *executable commands* are located in one of about four directories. A file is located in the directory

[12] Ability to evaporate or become chaotic.
[13] As usual, meaning a directory and all its subdirectories and all the files in those subdirectories etc.

tree according to its type, and not according to what software package it belongs to. Hence, for example, a word processor may have its actual executable stored in a directory with all other executables, while its font files are stored in a director with other fonts from all other packages.

The shell has a procedure for searching for executables when you type them in. If you type in a command with slashes, like `/bin/cp` then it tries to run the named program: `cp` out of the `/bin` directory. If you just type `cp` on its own, then it tries to find the `cp` command in each of the subdirectories of your `PATH`. To see what your `PATH` is, just type

```
echo $PATH
```

You will see a colon separated list of four or more directories. Note that the current directory `.` is not listed. It is very important that the current directory **not** be listed for security reasons. To execute a command in the current directory, we hence always type `./<command>`.

To append for example a new directory `/opt/gnome/bin` to your `PATH`, do

```
PATH="$PATH:/opt/gnome/bin"
export PATH
```

LINUX supports the convenience of doing this in one line:

```
export PATH="$PATH:/opt/gnome/bin"
```

Chapter 5

Regular Expressions

5.1 Basic regular expression exposition

A regular expression is a sequence of characters that forms a template used to search for *strings*[1] within text. To get an idea of when one would need to do this, consider the example where you have a list of names and telephone numbers. If you would like to find a telephone number that contains a 3 in the second place and ends with an 8, then regular expressions provide a way of doing these kinds of searches. Also consider where you would like to send an email to fifty people, but replacing the word after the "Dear" with their own name to make the letter more personal. Regular expressions allow for this type of searching and replacing. Many utilities use the regular expression to give them greater power when manipulating text. The `grep` command is an example. Previously you used the `grep` command to locate only simple letter sequences in text. Now we will use it to search for regular expressions.

In the previous chapter you learned that the ? character can be used to signify that any character can take its place. This is said to be a *wildcard* and works with filenames. With regular expressions, the wildcard to use is the `.` character. So you can use the command `grep .3....8 <filename>` to find the seven character telephone number that you are looking for in the above example.

Regular expressions are used for line by line searches. For instance, if the seven characters were spread over two lines (i.e. they had a line break in the middle), then `grep` wouldn't find them. In general a program that uses regular expressions will consider searches one line at a time.

Here are some examples that will teach you the regular expression basics. We will use the `grep` command to show the use of regular expressions (remember that the `-w` option matches whole words only). here the expression itself will be enclosed in ' quotes for reasons which will be explained later.

`grep -w 't[a-i]e'` Matches the words `tee`, `the` and `tie`. The brackets have a special significance. They mean to match one characters that can be anything from `a` to `i`.

`grep -w 't[i-z]e'` Matches the words `tie` and `toe`.

`grep -w 'cr[a-m]*t'` Matches the words `credit`, `craft`, `credit` and `cricket`. The `*` means to match any number of the previous character, which in this case is any character from `a` through `u`.

`grep -w 'kr.*n'` Matches the words `kremlin` and `krypton`, because the `.` matches any character and the `*` means to match the dot any number of times.

`egrep -w '(th|sh).*rt'` Matches the words `shirt`, `short`, and `thwart`. The `|` means to match either the `th` or the `sh`. `egrep` is just like `grep` but supports *extended regular expressions* which allow for the `|` feature[2].

`grep -w 'thr[aeiou]*t'` Matches the words `threat` and `throat`. As you can see, a list of possible characters may be placed inside the square brackets.

[1] Words, phrases, or just about sequence of characters.

[2] The `|` character often denotes a logical `OR`, meaning that either the thing on the left or the right of the `|` is applicable. This is true of many programming languages.

`grep -w 'thr[^a-f]*t'` Matches the words `throughput` and `thrust`. The ^ after the first bracket means to match *any* character *except* the characters listed. Hence, for example, the word `thrift` is not matched because it contains an `f`.

The above regular expressions all match whole words (because of the `-w` option). If the `-w` option was not present they might match parts of words which would result in a far greater number of matches. Also note that although the * means to match any number of characters, it also will match **no** characters as well, for example: `t[a-i]*e` could actually match the letter sequence `te`. i.e a `t` and an `e` with zero characters between them.

Usually, you will use regular expression to search for *whole lines* that match, and sometimes you would like to match a line that begins or ends with a certain string. The `^` character is used to specify the beginning of a line and the `$` character for the end of the line. For example `^The` matches all lines that start with a `The`, while `hack$` matches all lines that end with `hack`, and `'^ *The.*hack *$'` matches all lines that begin with `The` and end with `hack`, even if there is whitespace at the beginning or end of the line.

Because regular expression use certain characters in a special way (these are `. \ [ ] * + ?`), these characters cannot be used to match characters. This provides a severe limitation when trying to match, say, file-names which often use the `.` character. To match a `.` you can use the sequence `\.` which forces interpretation as an actual `.` and not as a wildcard. Hence the regular expression `myfile.txt` might match the letter sequence `myfileqtxt` or `myfile.txt`, but the regular expression `myfile\.txt` will match only `myfile.txt`.

Most special characters can be specified by adding a \ character before them, eg use `\[` for an actual `[`, a `\$` for an actual `$`, a `\\` for and actual `\`, `\+` for an actual `+`, and `\?` for an actual `?`. (`?` and `+` are explained below.)

5.2 The `fgrep` command

`fgrep` is an alternative to `grep`. The difference is that while `grep` (the more commonly used command) matches regular expressions, `fgrep` matches literal strings. In other words you can use `fgrep` where you would like to search for an ordinary string that is not a regular expression, instead of preceding special characters with `\`.

5.3 Regular expression `\{ \}` notation

`x*` matches zero to infinite instances of a character `x`. You can specify other ranges of numbers of characters to be matched with, for example `x\{3,5\}`, which will match at least three, but not more than five `x`'s, that is `xxx`, `xxxx`, or `xxxx`.

`x\{4\}`, can then be used to match 4 `x`'s exactly and no more and no less. `x\{7,\}` will match seven or more `x`'s — the upper limit is omitted to mean that there is no maximum number of `x`'s.

As in all the examples above, the x can be a range of characters (like `[a-k]`) just as well as a single charcter.

`grep -w 'th[a-t]\{2,3\}t'` Matches the words `theft`, `thirst`, `threat`, `thrift` and `throat`.

`grep -w 'th[a-t]\{4,5\}t'` Matches the words `theorist`, `thicket` and `thinnest`.

5.4 Extended regular expression `+ ? \< \> ( ) |` notation with `egrep`

There is an enhanced version of regular expressions that allows for a few more useful features. Where these conflict with existing notation, they are only available through the `egrep` command.

`+` is analogous to `\{1,\}`, it does the same as `*`, but matches *one* or more character, instead of *zero* or more characters.

`?` is analogous to \{1\}, it matches *zero* or *one* characters.

`< >` can surround a string to match only whole words.

`( )` can surround several strings, seperated by `|`. This will match any of these strings.

The following examples should make the last two notations clearer.

`grep 'trot'` Matches the words `electrotherapist`, `betroth` and so on, but

`grep '\<trot\>'` Matches only the word `trot`.

`egrep -w '(this|that|c[aeiou]*t)'` Matches the words `this`, `that`, `cot`, `coat`, `cat` and `cut`.

5.5 Regular expression sub-exressions

These are covered in the chapter, *Streams and sed - the stream editor*.

Chapter 6

Shell Scripting

6.1 Introduction

This chapter will introduce your to the concept of *computer programming*. So far, you have entered commands one at a time. Computer programming is merely the idea of getting a number of commands to be executed, that in combination do some unique powerful function.

To see a number of commands get executed in sequence, create a file with a `.sh` extension, into which you will enter your commands. The `.sh` extension is not strictly necessary, but serves as a reminder that the file contains special text called a *shell script*. From now on, the word *script* will be used to describe and sequence of commands placed in a text file. Now do a,

```
chmod 0755 myfile.sh
```

which allow the file to be run in the explained way.

Edit the file. The first line should be as follows with no whitespace[1].

```
#!/bin/sh
```

which dictates that the following program is a *shell* script, meaning that it accepts the same sort of commands that you have normally been typing at the prompt. Now enter a number of commands that you would like to be executed. You can start with

```
echo "Hi there"
echo "what is your name? (Type your name here and press Enter)"
read NM
echo "Hello $NM"
```

Now exit out of your editor and type `./myfile.sh`. This will *execute*[2] the file. Note that typing `./myfile.sh` is no different from typing any other command at the shell prompt. Your file `myfile.sh` has in fact become a new UNIX command all of its own.

Now note what the `read` command is doing. It creates a pigeon-hole called `NM`, and then inserts text read from tjhe keyboard into that pigeon hole. Thereafter, whenever the shell encounters `NM`, its contents are written out instead of the letters NM (provided you write a `$` in front of it). We say that `NM` is a *variable* because its contents can vary.

You can use shell scripts like a calculator. Try,

```
echo "I will work out X*Y"
echo "Enter X"
read X
```

[1] Whitespace are tabs and spaces, and in some contexts, newline (end of line) characters.
[2] Cause the computer to read and act on your list of commands, also called *running* the program.

```
echo "Enter Y"
read Y
echo "X*Y = $X*$Y = $[X*Y]"
```

The [and] mean that everything between must be *evaluated*[3] as a *numerical expression*[4]. You can in fact do a calculation at any time by typing it at the prompt:

```
echo $[3*6+2*8+9]
```

```
5
```

6.2 Looping to repeat commands: the while statement

The shell reads each line in succession from top to bottom: this is called *program flow*. Now suppose you would like a command to be executed more than once — you would like to alter the program flow so that the shell reads particular commands repeatedly. The while command executes a sequence of commands many times. Here is an example (-le stands for *less than or equal*):

```
N=1
while test "$N" -le "10"
do
        echo "Number $N"
        N=$[N+1]
done
```

The N=1 creates a variable called N and places the number 1 into it. The while command executes all the commands between the do and the done repetatively until the "test" condition is no longer true (i.e until N is greater than 10). The -le stands for *-less-than-or-equal-to*. Do a man test to see the other types of tests you can do on variables. Also be aware of how N is replaced with a new value that becomes 1 greater with each repetition of the while loop.

You should note here that each line is distinct command — the commands are *newline-seperated*. You can also have more than one command on a line by seperating them with a semicolon as follows:

```
N=1 ; while test "$N" -le "10"; do echo "Number $N"; N=$[N+1] ; done
```

(Try counting down from 10 with -ge (*-greater-than-or-equal*).) It is easy to see that shell scripts are extremely powerful, because any kind of command can be executed with conditions and loops.

6.3 Looping to repeat commands: the for statement

The for command also allows execution of commands multiple times. It works like this::

```
for i in cows sheep chickens pigs
do
        echo "$i is a farm animal"
done
echo -e "but\nGNUs are not farm animals"
```

The for command takes each string after the in, and executes the lines between do and done with i substituted for that string. The strings can be anything (even numbers) but are often filenames.

The if command executes a number of commands if a condition is met (-gt stands for *greater than*, -lt stands for *less than*).

[3] Substituted, worked-out, or reduced to some simplified form.
[4] Sequence of numbers with +, -, * etc. between them.
[5] Note that the Bash shell that you are using allows such [ ] notation. On some UNIX systems you will have to use the expr command to get the same effect.

```
X=10
Y=5
if test "$X" -gt "$Y" ; then
        echo "$X is greater than $Y"
fi
```

The `if` command in its full form can contain as much as,

```
X=10
Y=5
if test "$X" -gt "$Y" ; then
        echo "$X is greater than $Y"
elif test "$X" -lt "$Y" ; then
        echo "$X is less than $Y"
else
        echo "$X is equal to $Y"
fi
```

Now let us create a script that interprets its arguments. Create a new script called `backup-lots.sh`, containing:

```
\#!/bin/sh
for i in 0 1 2 3 4 5 6 7 8 9 ; do
        cp $1 $1.BAK-$i
done
```

Now create a file `important_data` with anything in it and then run `./backup-lots.sh important_data`, which will copy the file ten times with ten different extensions. As you can see the variable `$1` has a special meaning — it is the first argument on the command line. Now lets get a little bit more sophisticated (`-e` test if the file *exists*):

```
#!/bin/sh
if test "$1" = "" ; then
        echo "Usage: backup-lots.sh <filename>"
        exit
fi
for i in 0 1 2 3 4 5 6 7 8 9 ; do
        NEW_FILE=$1.BAK-$i
        if test -e $NEW_FILE ; then
                echo "backup-lots.sh: ***warning*** $NEW_FILE"
                echo "                already exists -skipping"
        else
                cp $1 $NEW_FILE
        fi
done
```

6.4 Looping over glob expressions

We know that the shell can expand file names when given *wildcards*. For instance, we can type `ls *.txt` to list all files ending with `.txt`. This applies equally well in any situation: for instance:

```
#!/bin/sh
for i in *.txt ; do
        echo "found a file:" $i
done
```

The `*.txt` is expanded to all matching files. **These files are searched for in the current directory**. If you include an absolute path then the shell will search in that directory:

```
#!/bin/sh
for i in /usr/doc/*/*.txt ; do
        echo "found a file:" $i
done
```

Which demonstrates the shells ability to search for matching files and expand to an absolute path.

6.5 The case statement

The case statement can make a potentially complicated program very short. It is best explained with an example.

```
#!/bin/sh
case $1 in
        -{}-test|-t)
                echo "you used the -{}-test option"
                exit 0
        ;;
        -{}-help|-h)
                echo "Usage:"
                echo "        myprog.sh {[}-{}-test|-{}-help|-{}-version{]}"
                exit 0
        ;;
        -{}-version|-v)
                echo "myprog.sh version 0.0.1"
                exit 0
        ;;
        -*)
                echo "No such option $1"
                echo "Usage:"
                echo "        myprog.sh {[}-{}-test|-{}-help|-{}-version{]}"
                exit 1
        ;;
esac

echo "You typed \"$1\" on the command line"
```

Above you can see that we are trying to process the first argument to a program. It can be one of several options, so using if statements will come to a long program. The case statement allows us to specify several possible statement blocks depending on the value of a variable. Note how each statement block is seperated by ;;. The strings before the) are glob expression matches. The first successful match causes that block to be executed. The | symbol allows us to enter several possible glob expressions.

6.6 Using functions: the function keyword

So far our programs execute mostly from top to bottom. Often, code needs to be repeated, but it is considered bad programming practice to repeat groups of statements that have the same functionality. Function definitions provide a way to group statement blocks into one. A function groups a list of commands and assigns it a name. For example:

```
#!/bin/sh

function usage ()
{
        echo "Usage:"
        echo "        myprog.sh [--test|--help|--version]"
}

case $1 in
```

```
        --test|-t)
                echo "you used the --test option"
                exit 0
        ;;
        --help|-h)
                usage
        ;;
        --version|-v)
                echo "myprog.sh version 0.0.2"
                exit 0
        ;;
        -*)
                echo "Error: no such option $1"
                usage
                exit 1
        ;;
esac

echo "You typed \"$1\" on the command line"
```

Whereever the `usage` keyword appears, it is effectively substituted for the two lines inside the { and }. There are obvious advantages to this approach: if you would like to change the program *usage* description, you only need to change it in one place in the code. Good programs use functions so liberally that they never have more than 50 lines of program code in a row.

6.7 Properly processing command line arguments: the `shift` keyword

Most programs we have seen can take many command-line arguments in any order. Here is how we can make our own shell scripts with this functionality. The command-line arguments can be reached with `$1`, `$2`, etc. the script,

```
#!/bin/sh

echo "The first argument is: $1, second argument is: $2, third argument is: $3"
```

Can be run with,

```
myfile.sh dogs cats birds
```

and prints,

```
The first argument is: dogs, second argument is: cats, third argument is: birds
```

Now we need to loop through each argument and decide what to do with it. A script like

```
for i in $1 $2 $3 $4 ; do
        <statments>
done
```

doesn't give us much flexibilty. The `shift` keyword is meant to make things easier. It shifts up all the arguments by one place so that `$1` gets the value of `$2`, `$2` gets the value of `$3` and so on. (`!=` tests if the `"$1"` is not equal to `""`, i.e. if it is empty and is hence past the last argument.) Try:

```
while test "$1" != "" ; do
        echo $1
        shift
done
```

and run the program with lots of arguments. Now we can put any sort of condition statements within the loop to process the arguments in turn:

```
#!/bin/sh

function usage ()
{
        echo "Usage:"
        echo "         myprog.sh [--test|--help|--version] [--echo <text>]"
}

while test "$1" != "" ; do
         case $1 in
                 --echo|-e)
                         echo "$2"
                         shift
                 ;;
                 --test|-t)
                         echo "you used the --test option"
                 ;;
                 --help|-h)
                         usage
                         exit 0
                 ;;
                 --version|-v)
                         echo "myprog.sh version 0.0.3"
                         exit 0
                 ;;
                 -*)
                         echo "Error: no such option $1"
                         usage
                         exit 1
                 ;;
         esac
         shift
done
```

myprog.sh can now run with multiple arguments on the command-line.

6.8 More on command-line arguments: $@ and $0

Whereas $1, $2, $3 etc.expand to the individual arguments passed to the program, $@ expands to **all** arguments. This is useful for passing all remaining arguments onto a second command. For instance,

```
if test "$1" = "--special" ;
then
         shift
         myprog2.sh $@
fi
```

$0 means the name of the program itself and not any command line argument. It is the command used to envoke the current program. In the above cases it will be ./myprog.sh. Note that $0 is immune to shift's.

6.9 Single forward quote notation

Single forward quotes ' **protect** the enclosed text from the shell. In other words, you can place any odd characters inside forward quotes and the shell will treat them literally and reproduce your text exactly. For instance you may want to echo an actual $ to the screen to produce an output like costs $1000. You can use echo 'costs $1000' instead of echo "costs $1000".

6.10 Double quote notation

Double quotes " have the oposite sence to single quotes. They allow **all** shell interpretations to take place inside them. The reason they are used at all is only to group text containing white-space into a single word, becuase the shell will usually break up text along whitespace boundaries. Try `for i in "henry john mary sue" ; do echo "$i is a person" ; done`, compared to `for i in henry john mary sue ; do echo $i is a person ; done`.

6.11 Backward quote substitution

Backward quote ‘ have a special meaning to the shell. When a command is inside backward quotes it means that the command should be run and its **output** substituted in place of the backquotes. Take for example the `cat` command. Create a small file `to_be_catted` with only the text `daisy` inside it. Create a shell script

```
X=‘cat to_be_catted‘
echo $X
```

The value of `X` is set to the output of the cat command, which in this case is the word `daisy`. The is a powerful tool. Consider the `expr` command:

```
X=‘expr 100 + 50 ’*’ 3‘
echo $X
```

Hence we can use `expr` and backquotes to do mathematics inside our shell script. Here is function to calculate factorials. Note how we enclose the `*` in forward quotes. This is to prevent the shell from thinking that we want it to expand the `*` into matching file-names:

```
function factorial ()
{
    N=$1
    A=1
    while test $N -gt 0 ; do
        A=‘expr $A ’*’ $N‘
        N=‘expr $N - 1‘
    done
    echo $A
}
```

We can see that the square braces used further above can actually suffice for most of the times where we would like to use `expr`. (However `$[]` notation is an extension of the GNU shells, and is not a standard feature on all varients of UNIX.) We can now run `factorial 20` and see the output. If we would like to assign the output to a variable, this can be done with, `X=‘factorial 20‘`.

Chapter 7

Streams and sed — the stream editor

7.1 Introduction

[1]The commands grep, echo, df and so on print some output to the screen. In fact, what is happening on a lower level is that they are printing characters one by one into a theoretical data *stream* (also called a *pipe*) called the *stdout* pipe. The shell itself performs the action of reading those characters one by one and displaying them on the screen. The word *pipe* itself means exactly that: a program places data in the one end of a funnel while another program reads that data from the other end. The reason for pipes is to allow two seperate programs to perform simple communications with each other. In this case, the program is merely communicating with the shell in order to display some output.

The same is true with the cat command explained previously. This command run with no arguments reads from the *stdin* pipe. By default this is the keyboard. One further pipe is the *stderr* pipe which a program writes error messages to. It is not possible to see whether a program message is caused by the program writing to its stderr or stdout pipe, because usually both are directed to the screen. Good programs however always write to the appropriate pipes to allow output to be specially seperated for diagnostic purposes if need be.

7.2 Tutorial

Create a text file with lots of lines that contain the word GNU and one line that contains the word GNU as well the word Linux. Then do grep GNU myfile.txt. The result is printed to stdout as usual. Now try grep GNU myfile.txt > gnu_lines.txt. What is happening here is that the output of the grep command is being *redirected* into a file. The > gnu_lines.txt tells the shell to create a new file gnu_lines.txt and fill it with any output from stdout, instead of displaying the output as it usually does. If the file already exists, it will be *truncated*[2].

Now suppose you want to append further output to this file. Using >> instead of > will **not** truncate the file but append any output to it. Try this: echo "morestuff" >> gnu_lines.txt. Then view the contents of gnu_lines.txt.

7.3 Piping using | notation

The real power of pipes is when one program can read from the output of another program. Consider the grep command which reads from stdin when given no arguments: run grep with one argument on the command line:

```
# grep GNU
A line without that word in it
```

[1]The ability to use pipes is one of the powers of UNIX. This is one of the principle deficiancies of some non-UNIX systems. Pipes used on the command line as explained in this section are a neat trick, but Pipes used inside C programs enormously simplify program interaction. Without pipes, huge amounts of complex and buggy code usually needs to be written to perform simple tasks. It is hoped that these ideas will give the reader an idea of why UNIX is such a ubiquitous and enduring standard.

[2]Shortened to zero length

```
Another line without that word in it
A line with the word GNU in it
A line with the word GNU in it
I have the idea now
^C
#
```

grep's default is to read from stdin when no files are given. As you can see, it is doing its usual work of printing out lines that have the word GNU in them. Hence lines containing GNU will be printed twice - as you type them in and again when grep reads them and decides that they contain GNU.

Now try grep GNU myfile.txt | grep Linux. The first grep outputs all lines with the word GNU in them to stdout. The | tells that all stdout is to be typed as stdin (us we just did above) into the next command, which is also a grep command. The second grep command scans that data for lines with the word Linux in them. grep is often used this way as a *filter*[3] and be used multiple times eg. grep L myfile.txt | grep i | grep n | grep u | grep x.

7.4 A complex piping example

In a previous chapter we used grep on a dictionary to demonstrate regular expressions. This is how a dictionary of words can be created:

```
cat /usr/lib/ispell/english.hash | strings | tr 'A-Z' 'a-z' \
| grep '^[a-z]' | sort -u > mydict
```

[4] The file english.hash contains the UNIX dictionary normally used for spell checking. With a bit of filtering you can create a dictionary that will make solving crossword puzzles a breese. First we use the command strings explained previously to extract readable bits of text. Here we are using its alternate mode of operation where it reads from stdin when no files are specified on its command-line. The command tr (abbreviated from *translate* see the tr man page.) then converts upper to lower case. The grep command then filters out lines that do not start with a letter. Finally the sort command sorts the words in alphabetical order. The -u option stands for unique, and specifies that there should be not duplicate lines of text. Now try less mydict.

7.5 Redirecting streams with >&

Try the command ls nofile.txt > A. ls should give an error message if the file doesn't exist. The error message is however displayed, and not written into the file A. This is because ls has written its error message to stderr while > has only redirected stdout. The way to get both stdout and stderr to both go to the same file is to use a *redirection operator*. As far as the shell is concerned, stdout is called 1 and stderr is called 2, and commands can be appended with a *redirection* like 2>&1 to dictate that stderr is to be mixed into the output of stdout. The actual words stderr and stdout are only used in C programming. Try the following:

```
touch existing_file
rm -f non-existing_file
ls existing_file non-existing_file
```

ls will output two lines: a line containing a listing for the file existing_file and a line containing an error message to explain that the file non-existing_file does not exist. The error message would have been written to stderr or *file descriptor* number 2, and the remaining line would have been written to stdout or *file descriptor* number 1. Next we try

```
ls existing_file non-existing_file 2>A
cat A
```

[3] Something that screens data.

[4] A backslash \ as the last character on a line indicates that the line is to be continued. You can leave out the \but then you must leave out the newline as well.

Now A contains the error message, while the remaining output came to the screen. Now try,

```
ls existing_file non-existing_file 1>A
cat A
```

The notation 1>A is the same as >A because the shell assumes that you are referring to file descriptor 1 when you don't specify any. Now A contains the stdout output, while the error message has been redirected to the screen. Now try,

```
ls existing_file non-existing_file 1>A 2>&1
cat A
```

Now A contains both the error message and the normal output. The >& is called a *redirection operator*. x>&y tells to write pipe x into pipe y. **Redirection is specified from right too left on the command line**. Hence the above command means to mix stderr into stdout and **then** to redirect stdout to the file A. Finally,

```
ls existing_file non-existing_file 2>A 1>&2
cat A
```

We notice that this has the same effect, except that here we are doing the reverse: redirecting stdout into stderr, and then redirecting stderr into a file A. To see what happens if we redirect in reverse order, we can try,

```
ls existing_file non-existing_file 2>&1 1>A
cat A
```

which means to redirect stdout into a file A, and **then** to redirect stderr into stdout. This will therefore not mix stderr and stdout because the redirection to A came first.

7.6 Using sed to edit streams

ed used to be the standard text *ed*itor for UNIX. It is cryptic to use, but is compact and programmable. sed stands for *stream editor*, and is the only incarnation of ed that is commonly used today. sed allows editing of files non-interactively. In the way that grep can search for words and filter lines of text; *sed* can do search-replace operations and insert and delete lines into text files. sed is one of those programs with no man page to speek of. Do info sed to see sed's comprehensive info pages with examples. The most common usage of sed is to replace words in a stream with alternative words. *sed* reads from stdin and writes to stdout. Like grep, it is line buffered which means that it reads one line in at a time and then writes that line out again after performing whatever editing operations. Replacements are typically done with:

```
cat <file> | sed -e 's/<search-regexp>/<replace-text>/<option>' \
> <resultfile>
```

where *search-regexp* is a regular expression, *replace-text* is the text you would like to replace each found occurance with, and *option* is nothing or g, which means to replace every occurance in the same line (usually sed just replaces the first occurance of the regular expression in each line). (There are other *option*s, see the sed info page.) For demonstration, type

```
sed -e 's/e/E/g'
```

and type out a few lines of english text.

sed is actually an extremely powerful and important system of editing. A complete overview will be done later. Here we will concentrate on searching and replacing regular expressions.

7.7 Regular expression sub-exressions

The section explains how to do the apparently complex task of moving text around within lines. Consider for example the output of `ls`: now say you want to automatically strip out only the size column — `sed` can do this sort of editing using the special `\( \)` notation to group parts of the regular expression together. Consider the following example:

```
sed -e 's/\(\<[^ ]*\>\)\([ ]*\)\(\<[^ ]*\>\)/\3\2\1/g'
```

Here `sed` is searching for the expression `\<.*\>[ ]*\<.*\>`. From the chapter on regular expressions, we can see that it matches a whole word, an arbitrary amount of whitespace, and then another whole word. The `\( \)` groups these three so that they can be referred to in *replace-text*. Each part of the regular expression inside `\( \)` is called a *sub-expression* of the regular expresion. Each sub-expression is numbered — namely `\1`, `\2` etc. Hence `\1` in *replace-text* is the first `\<[^ ]*\>`, `\2` is `[ ]*`, and finally, `\3` is the second `\<[^ ]*\>`. Now test to see what happens when you run this:

```
sed -e 's/\(\<[^ ]*\>\)\([ ]*\)\(\<[^ ]*\>\)/\3\2\1/g'
GNU Linux is cool
Linux GNU cool is
```

To return to our `ls` example (note that this is just an example, to count file sizes you should rather use the `du` command), think about if we would like to sum the bytes sizes of all the files in a directory:

```
expr 0 `ls -l | grep '^-' | \
  sed 's/^\([^ ]*[ ]*\)\{4,4\}\([0-9]*\).*$/ + \2/'`
```

We know that `ls -l` output lines start with `-` for ordinary files. So we use `grep` to strip lines not starting with `-`. If we do an `ls -l`, we see the output is divided into four columns of stuff we are not interested in, and then a number indicating the size of the file. A column (or *field*) can be described by the regular expression `[^ ]*[ ]*`, i.e. a length of text with no whitespace, followed by a length of whitespace. There are four of these, so we bracket it with `\( \)`, and then use the `\{ \}` notation to indicate that we want exactly `4`. After that comes our number `[0-9]*`, and then any trailing characters which we are not interested in, `.*$`. Notice here that we have neglected to use `\< \>` notation to indicate whole words. This is because `sed` tries to match the maximum number of characters legally allowed, and in the situation we have here, has exactly the same effect.

If you haven't yet figured it out, we are trying to get that column of bytes sizes into the format like,

```
+ 438
+ 1525
+ 76
+ 92146
```

... so that `expr` can understand it. Hence we replace each line with sub-expression `\2` and a leading `+` sign. Backquotes give the output of this to `expr`, which sums them studiously, ignoring any newline characters as though the summation were typed in on a single line. There is one minor problem here: the first line contains a `+` with nothing before it, which will cause `expr` to complain. To get around this, we can just add a `0` to the expression, so that it becomes `0 +` ....

Chapter 8

Processes and environment variables

8.1 Introduction

On UNIX, when you run a program (like any of the shell commands you have been using) the actual computer instructions are read out of a file on disk out of one of the `/bin/` directories and placed in *RAM*[1]. The program then gets executed in memory and becomes a *process*. A *process* is some command/program/shell-script that is being run (or *executed*) in memory. When the process is finished running, it is removed from memory. There are usually about 50 processes running simultaneously at any one time on a system with one person logged in. The *CPU*[2] hops between each of them to give a share of its *execution time*[3].Each process is given a process number called the *PID* (Process ID). Besides the memory actually occupied by the process, the process itself ceases addition memory for its operations.

In the same way that a file is owned by a particular user and group, a process also has an owner — usually the person who ran the program. Whenever a process tries to access a file, its ownerships is compared to that of the file to decide if the access is permissable. Because all devices are files, the only way a process can do *anything* is through a file, and hence file permission restrictions are the only kind of restrictions there need ever be on UNIX. This is how UNIX security works.

8.2 Tutorial

Login on a terminal and type the command `ps`. You should get some output like:

```
 PID  TTY  STAT   TIME  COMMAND
5995    2  S     0:00  /bin/login  --  myname
5999    2  S     0:00  -bash
6030    2  R     0:00  ps
```

`ps` with no options shows 3 processes to be running. These are the only three processes visible to you as a user, although there are other system processes not belonging to you. The first process was the program that logged you in by displaying the login prompt and requesting a password. It then ran a second process call `bash`, the Bourne Again Shell[4] where you have been typing commands. Finally you ran `ps`, hence it must have found itself when it checked for what processed were running, but then exited immediately afterward.

8.2.1 Controlling jobs

The shell has many facilities for controlling and executing processes — this is called job control. Create a small script called `proc.sh`:

[1] Random Access Memory — the memory chips on your motherboard.
[2] Central Processing Unit — the actual 386/486 or whatever chip that sits on your motherboard.
[3] Time given to carry out the instructions of a particular program. Note this is in contrast to Windows or DOS where the program itself has to allow the others a share of the CPU: under UNIX, the process has no say in the matter.
[4] The Bourne shell was the original UNIX shell

```
#!/bin/sh
echo "proc.sh: is running"
sleep 1000
```

Run the script with `chmod 0755 proc.sh` and then `./proc.sh`. The shell will *block* waiting for the process to exit. Now hit ^Z[5]. This will *stop* the process. Now do a `ps` again. You will see your script listed. However it is not presently running because it is in the condition of being stopped. Type `bg` standing for *background*. The script will now be un-stopped and run in the background. You can now run other processes in the mean time. Type `fg` and the script will return to the *foreground*. You can then type ^C to interrupt the process.

Creating background processes

Create a program that does something a little more interesting:

```
#!/bin/sh
echo "proc.sh: is running"
while true ; do
        echo -e '\a'
        sleep 2
done
```

Now perform the ^Z, `bg`, `fg` and ^C operations from before. To put a process immediately into the background, you can use:

```
./proc.sh &
```

The **JOB CONTROL** section of the `bash` man page (`bash`(1)) looks like this:

> *Job control* refers to the ability to selectively stop (*suspend*) the execution of processes and continue (*resume*) their execution at a later point. A user typically employs this facility via an interactive interface supplied jointly by the system's terminal driver and **bash**.
>
> The shell associates a *job* with each pipeline. It keeps a table of currently executing jobs, which may be listed with the **jobs** command. When **bash** starts a job asynchronously (in the *background*), it prints a line that looks like:
>
> **[1] 25647**
>
> indicating that this job is job number 1 and that the process ID of the last process in the pipeline associated with this job is 25647. All of the processes in a single pipeline are members of the same job. **Bash** uses the *job* abstraction as the basis for job control.
>
> To facilitate the implementation of the user interface to job control, the system maintains the notion of a *current terminal process group ID*. Members of this process group (processes whose process group ID is equal to the current terminal process group ID) receive keyboard-generated signals such as **SIGINT**. These processes are said to be in the *foreground*. *Background* processes are those whose process group ID differs from the terminal's; such processes are immune to keyboard-generated signals. Only foreground processes are allowed to read from or write to the terminal. Background processes which attempt to read from (write to) the terminal are sent a **SIGTTIN (SIGTTOU)** signal by the terminal driver, which, unless caught, suspends the process.
>
> If the operating system on which **bash** is running supports job control, **bash** allows you to use it. Typing the *suspend* character (typically **^Z**, Control-Z) while a process is running causes that process to be stopped and returns you to **bash**. Typing the *delayed suspend* character (typically **^Y**, Control-Y) causes the process to be stopped when it attempts to read input from the terminal,

[5] ^ means to hold down the Ctrl key and press the Z key.

and control to be returned to **bash**. You may then manipulate the state of this job, using the **bg** command to continue it in the background, the **fg** command to continue it in the foreground, or the **kill** command to kill it. A ^**Z** takes effect immediately, and has the additional side effect of causing pending output and typeahead to be discarded.

There are a number of ways to refer to a job in the shell. The character % introduces a job name. Job number *n* may be referred to as **%n**. A job may also be referred to using a prefix of the name used to start it, or using a substring that appears in its command line. For example, **%ce** refers to a stopped **ce** job. If a prefix matches more than one job, **bash** reports an error. Using **%?ce**, on the other hand, refers to any job containing the string **ce** in its command line. If the substring matches more than one job, **bash** reports an error. The symbols %% and %+ refer to the shell's notion of the *current job*, which is the last job stopped while it was in the foreground. The *previous job* may be referenced using %-. In output pertaining to jobs (e.g., the output of the **jobs** command), the current job is always flagged with a +, and the previous job with a -.

Simply naming a job can be used to bring it into the foreground: **%1** is a synonym for **"fg %1"**, bringing job 1 from the background into the foreground. Similarly, **"%1 &"** resumes job 1 in the background, equivalent to **"bg %1"**.

The shell learns immediately whenever a job changes state. Normally, **bash** waits until it is about to print a prompt before reporting changes in a job's status so as to not interrupt any other output. If the **-b** option to the **set** builtin command is set, **bash** reports such changes immediately. (See also the description of **notify** variable under **Shell Variables** above.)

If you attempt to exit **bash** while jobs are stopped, the shell prints a message warning you. You may then use the **jobs** command to inspect their status. If you do this, or try to exit again immediately, you are not warned again, and the stopped jobs are terminated.

8.2.2 `kill`*ing* a process, sending a process a signal

To terminate a process, use the `kill` command:

```
kill <PID>
```

The `kill` command actually sends a signal to the process causing it to execute some function. In some cases, the developers would not have bothered to account for this signal and some default behaviour happens.

To send a signal to a process you can name the signal on the command-line or use its numerical equivalent:

```
kill -SIGTERM 12345
```

or

```
kill -15 12345
```

Which is the signal that `kill` normally sends: the *termination* signal.

To unconditionally terminate a process:

```
kill -SIGKILL 12345
```

or

```
kill -9 12345
```

Which should only be used as a last resort.

It is cumbersome to have to constantly look up the PID of a process. Hence the GNU utilities have a command `killall` which sends a signal to all processes of the same name:

```
killall -<signal> <process_name>
```

This is useful when you are sure that there is only one of a process running, either because there is no one else logged in on the system, or because you are not logged in as super user.

The list of signals can be gotten from signal(7).

8.2.3 List of comman signals

SIGHUP *Hang up.* If the terminal becomes disconnected from a process, this signal is sent automatically to the process. Sending a process this signal often causes it to reread its configuration files, so it is useful instead of restarting the process. Always check the man page to see if a process has this behaviour.

SIGINT *Interrupt* from keyboard. Issued if you press ^C.

SIGQUIT *Quit* from keyboard. Issued if you press ^D.

SIGFPE *Floating Point Exception.* Issued automatically to a program performing some kind of illegal mathematical operation.

SIGKILL *Kill* Signal. This is one of the signals that can never be *caught* by a process. If a process gets this signal it **has** to quit immediately and will not perform any clean-up operations (like closing files or removing temporary files). You can send a process a SIGKILL signal if there is no other means of destroying it.

SIGSEGV *Segmentation Violation.* Issued automatically when a process tries to access memory outside of its allowable address space. I.e. equivalent to a **Fatal Exception** under Windows. Note that programs with bugs or programs in the process of being developed often get these. A program receiving a SIGSEGV however can never cause the rest of the system to be compromised. If the kernel itself were to receive such an error, it would cause the system to come down, but such is extremely rare.

SIGPIPE *Pipe* died. A program was writing to a pipe, the other end of which is no longer available.

SIGTERM *Terminate.* Cause the program to quit gracefully

8.2.4 *Environment*'s of processes

Each process that runs does so with the knowledge of several *var=value* text pairs. All this means is that a process can look up the value of some variable that it may have inherited from its parent process. The complete list of these text pairs is called the *environment* of the process, and each *var* is called an *environment variable*. Each process has its own environment, which is copied from the parent processes environment.

After you have logged in and have a shell prompt, the process you are using (the shell itself) is just like any other process with an environment with environment variables. To get a complete list of these variables, just type:

```
set
```

This is useful to find the value of an environment variable whose name you are unsure of:

```
set | grep <regexp>
```

Try `set | grep PATH` to see the PATH environment variable discussed previously.

The purpose of an environment is just to have an alternative way of passing parameters to a program (in addition to command-line arguments). The difference is that an environment is inherited from one process to the next: i.e. a shell might have certain variable set (like the PATH) and may run a file manager which may run a word-processor. The word-processor inherited its environment from file-manager which inherited its environment from the shell.

Try

```
X="Hi there!"
echo $X
```

You have set a variable. But now run

```
bash
```

You have now run a new process which is a *child* of the process you were just in. Type

```
echo $X
```

You will see that `X` is not set. This is because the variable was not `export`*ed* as an environment variable, and hence was not inherited. Now type

```
exit
```

Which returns you to the *parent* process. Then

```
export X
bash
echo $X
```

You will see that the new `bash` now knows about `X`.

Above we are setting an arbitrary variable for our own use. `bash` (and many other programs) automatically set many of their own environment variables. The `bash` man page lists these (when it talks about *unset*ting a variable, it means using the command `unset <variable>`). You may not understand some of these at the moment, but they are included here as a complete reference for later:

Shell Variables

The following variables are set by the shell:

PPID The process ID of the shell's parent.

PWD The current working directory as set by the **cd** command.

OLDPWD The previous working directory as set by the **cd** command.

REPLY Set to the line of input read by the **read** builtin command when no arguments are supplied.

UID Expands to the user ID of the current user, initialized at shell startup.

EUID Expands to the effective user ID of the current user, initialized at shell startup.

BASH Expands to the full pathname used to invoke this instance of **bash**.

BASH_VERSION Expands to the version number of this instance of **bash**.

SHLVL Incremented by one each time an instance of **bash** is started.

RANDOM Each time this parameter is referenced, a random integer is generated. The sequence of random numbers may be initialized by assigning a value to **RANDOM**. If **RANDOM** is unset, it loses its special properties, even if it is subsequently reset.

SECONDS Each time this parameter is referenced, the number of seconds since shell invocation is returned. If a value is assigned to **SECONDS**. the value returned upon subsequent references is the number of seconds since the assignment plus the value assigned. If **SECONDS** is unset, it loses its special properties, even if it is subsequently reset.

LINENO Each time this parameter is referenced, the shell substitutes a decimal number representing the current sequential line number (starting with 1) within a script or function. When not in a script or function, the value substituted is not guaranteed to be meaningful. When in a function, the value is not the number of the source line that the command appears on (that information has been lost by the time the function is executed), but is an approximation of the number of *simple commands* executed in the current function. If **LINENO** is unset, it loses its special properties, even if it is subsequently reset.

HISTCMD The history number, or index in the history list, of the current command. If **HISTCMD** is unset, it loses its special properties, even if it is subsequently reset.

OPTARG The value of the last option argument processed by the **getopts** builtin command (see **SHELL BUILTIN COMMANDS** below).

OPTIND The index of the next argument to be processed by the **getopts** builtin command (see **SHELL BUILTIN COMMANDS** below).

HOSTTYPE Automatically set to a string that uniquely describes the type of machine on which **bash** is executing. The default is system-dependent.

OSTYPE Automatically set to a string that describes the operating system on which **bash** is executing. The default is system-dependent.

There are also many variables that `bash` uses which may be set by the user. These are:

The following variables are used by the shell. In some cases, **bash** assigns a default value to a variable; these cases are noted below.

IFS The *Internal Field Separator* that is used for word splitting after expansion and to split lines into words with the **read** builtin command. The default value is "<space><tab><newline>".

PATH The search path for commands. It is a colon-separated list of directories in which the shell looks for commands (see **COMMAND EXECUTION** below). The default path is system-dependent, and is set by the administrator who installs **bash**. A common value is "/usr/gnu/bin:/usr/local/bin:/usr/ucb:/bin:/usr/bin:.".

HOME The home directory of the current user; the default argument for the **cd** builtin command.

CDPATH The search path for the **cd** command. This is a colon-separated list of directories in which the shell looks for destination directories specified by the **cd** command. A sample value is ``.:~:/usr''.

ENV If this parameter is set when **bash** is executing a shell script, its value is interpreted as a filename containing commands to initialize the shell, as in *.bashrc*. The value of **ENV** is subjected to parameter expansion, command substitution, and arithmetic expansion before being interpreted as a pathname. **PATH** is not used to search for the resultant pathname.

MAIL If this parameter is set to a filename and the **MAILPATH** variable is not set, **bash** informs the user of the arrival of mail in the specified file.

MAILCHECK Specifies how often (in seconds) **bash** checks for mail. The default is 60 seconds. When it is time to check for mail, the shell does so before prompting. If this variable is unset, the shell disables mail checking.

MAILPATH A colon-separated list of pathnames to be checked for mail. The message to be printed may be specified by separating the pathname from the message with a '?'. $_ stands for the name of the current mailfile. Example:
`MAILPATH='/usr/spool/mail/bfox?"You have mail":~/shell-mail?"$_ has mail!"'`
Bash supplies a default value for this variable, but the location of the user mail files that it uses is system dependent (e.g., /usr/spool/mail/**$USER**).

MAIL_WARNING If set, and a file that **bash** is checking for mail has been accessed since the last time it was checked, the message "The mail in *mailfile* has been read" is printed.

PS1 The value of this parameter is expanded (see **PROMPTING** below) and used as the primary prompt string. The default value is "**bash\$** ".

PS2 The value of this parameter is expanded and used as the secondary prompt string. The default is "> ".

PS3 The value of this parameter is used as the prompt for the *select* command (see **SHELL GRAMMAR** above).

PS4 The value of this parameter is expanded and the value is printed before each command **bash** displays during an execution trace. The first character of **PS4** is replicated multiple times, as necessary, to indicate multiple levels of indirection. The default is "+ ".

HISTSIZE The number of commands to remember in the command history (see **HISTORY** below). The default value is 500.

HISTFILE The name of the file in which command history is saved. (See **HISTORY** below.) The default value is *~/.bash_history*. If unset, the command history is not saved when an interactive shell exits.

HISTFILESIZE The maximum number of lines contained in the history file. When this variable is assigned a value, the history file is truncated, if necessary, to contain no more than that number of lines. The default value is 500.

OPTERR If set to the value 1, **bash** displays error messages generated by the **getopts** builtin command (see **SHELL BUILTIN COMMANDS** below). **OPTERR** is initialized to 1 each time the shell is invoked or a shell script is executed.

PROMPT_COMMAND If set, the value is executed as a command prior to issuing each primary prompt.

IGNOREEOF Controls the action of the shell on receipt of an **EOF** character as the sole input. If set, the value is the number of consecutive **EOF** characters typed as the first characters on an input line before **bash** exits. If the variable exists but does not have a numeric value, or has no value, the default value is 10. If it does not exist, **EOF** signifies the end of input to the shell. This is only in effect for interactive shells.

TMOUT If set to a value greater than zero, the value is interpreted as the number of seconds to wait for input after issuing the primary prompt. **Bash** terminates after waiting for that number of seconds if input does not arrive.

FCEDIT The default editor for the **fc** builtin command.

FIGNORE A colon-separated list of suffixes to ignore when performing filename completion (see **READLINE** below). A filename whose suffix matches one of the entries in **FIGNORE** is excluded from the list of matched filenames. A sample value is ".o:~".

INPUTRC The filename for the readline startup file, overriding the default of *~/.inputrc* (see **READLINE** below).

notify If set, **bash** reports terminated background jobs immediately, rather than waiting until before printing the next primary prompt (see also the **-b** option to the **set** builtin command).

history_control

HISTCONTROL If set to a value of *ignorespace*, lines which begin with a **space** character are not entered on the history list. If set to a value of *ignoredups*, lines matching the last history line are not entered. A value of *ignoreboth* combines the two options. If unset, or if set to any other value than those above, all lines read by the parser are saved on the history list.

command_oriented_history If set, **bash** attempts to save all lines of a multiple-line command in the same history entry. This allows easy re-editing of multi-line commands.

glob_dot_filenames If set, **bash** includes filenames beginning with a '.' in the results of pathname expansion.

allow_null_glob_expansion If set, **bash** allows pathname patterns which match no files (see **Pathname Expansion** below) to expand to a null string, rather than themselves.

histchars The two or three characters which control history expansion and tokenization (see **HISTORY EXPANSION** below). The first character is the *history expansion character*, that is, the character which signals the start of a history expansion, normally '**!**'. The second character is the *quick substitution* character, which is used as shorthand for re-running the previous command entered, substituting one string for another in the command. The default is '^'. The optional third character is the character which signifies that the remainder of the line is a comment, when found as the first character of a word, normally '#'. The history comment character causes history substitution to be skipped for the remaining words on the line. It does not necessarily cause the shell parser to treat the rest of the line as a comment.

nolinks If set, the shell does not follow symbolic links when executing commands that change the current working directory. It uses the physical directory structure instead. By default, **bash** follows the logical chain of directories when performing commands which change the current directory, such as **cd**. See also the description of the **-P** option to the **set** builtin (**SHELL BUILTIN COMMANDS** below).

hostname_completion_file

HOSTFILE Contains the name of a file in the same format as */etc/hosts* that should be read when the shell needs to complete a hostname. The file may be changed interactively; the next time hostname completion is attempted **bash** adds the contents of the new file to the already existing database.

noclobber If set, **bash** does not overwrite an existing file with the >, >&, and <> redirection operators. This variable may be overridden when creating output files by using the redirection operator >— instead of > (see also the **-C** option to the **set** builtin command).

auto_resume This variable controls how the shell interacts with the user and job control. If this variable is set, single word simple commands without redirections are treated as candidates for resumption of an existing stopped job. There is no ambiguity allowed; if there is more than one job beginning with the string typed, the job most recently accessed is selected. The *name* of a stopped job, in this context, is the command line used to start it. If set to the value *exact*, the string supplied must match the name of a stopped job exactly; if set to *substring*, the string supplied needs to match a substring of the name of a stopped job. The *substring* value provides functionality analogous to the **%?** job id (see **JOB CONTROL** below). If set to any other value, the supplied string must be a prefix of a stopped job's name; this provides functionality analogous to the **%** job id.

no_exit_on_failed_exec If this variable exists, a non-interactive shell will not exit if it cannot execute the file specified in the **exec** builtin command. An interactive shell does not exit if **exec** fails.

cdable_vars If this is set, an argument to the **cd** builtin command that is not a directory is assumed to be the name of a variable whose value is the directory to change to.

Chapter 9

Mail

Electronic Mail or *e*mail is the way most people first come into contact with the internet. Although you may have used email in a graphical environment, here we will show you how mail was first intended to be used on a multi-user system. To a large extent what applies here is really what is going on in the background of any system that supports mail.

A mail message is a block of text sent from one user to another using some mail command or mailer program. Although a mail message will usually be accompanied by a *subject* explaining what the mail is about. The idea of mail is that a message can be sent to someone even though he may not be logged in at the time and the mail will be stored for him until he is around to read it. An email address is probably familiar to you, such as: `jack@kangeroo.co.au`. This means that `jack` has a user account on a computer called `kangeroo.co.au`. The text after the `@` is always the name of the machine. Todays Internet does not obey this exactly, but there is always a machine that `jack` **does** have an account on where mail is eventually sent.

When mail is received for you (from another user on the system or from a user from another system) it is appended to the file `/var/spool/mail/<username>` called the *mail file* or *mail box file*. Where ¡username¿ is your login name. You then run some program which interprets your mail file, allowing you to browse the file as a sequence of mail messages and read and reply to them.

An actual addition to your mail file might look like this:

```
From maynard@iafrica.com  Mon Jun  1 21:20:21 1998
Return-Path: <maynard@iafrica.com>
Received: from lava.obsidian.co.za (root@lava.obsidian.co.za [192.168.2.254])
        by ra.obsidian.co.za (8.8.7/8.8.7) with ESMTP id VAA11942
        for <psheer@obsidian.co.za>; Mon, 1 Jun 1998 21:20:20 +0200
Received: from mail450.icon.co.za (mail450.icon.co.za [196.26.208.3])
        by lava.obsidian.co.za (8.8.5/8.8.5) with ESMTP id VAA19357
        for <psheer@obsidian.co.za>; Mon, 1 Jun 1998 21:17:06 +0200
Received: from smtp02.iafrica.com (smtp02.iafrica.com [196.7.0.140])
        by mail450.icon.co.za (8.8.8/8.8.8) with SMTP id VAA02315
        for <psheer@icon.co.za>; Mon, 1 Jun 1998 21:24:21 +0200 (GMT)
Received: from default [196.31.19.216] (fullmoon)
        by smtp02.iafrica.com with smtp (Exim 1.73 #1)
        id 0ygTDL-00041u-00; Mon, 1 Jun 1998 13:57:20 +0200
Message-ID: <357296DF.60A3@iafrica.com>
Date: Mon, 01 Jun 1998 13:56:15 +0200
From: a person <maynard@iafrica.com>
Reply-To: maynard@iafrica.com
Organization: private
X-Mailer: Mozilla 3.01 (Win95; I)
MIME-Version: 1.0
To: paul sheer <psheer@icon.co.za>
Subject: hello
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Status: RO
```

```
X-Status: A

hey paul
its me
how r u doing
i am well
what u been upot
hows life
hope your well
amanda
```

Each mail message begins with a `From` at the beginning of a line, followed by a space. Then comes the *mail header*, explaining where the message was routed from to get it to your mail box, who sent the message, where replies should go to, the subject of the mail, and various other fields. Above, the header is longer than the mail messages. Examine the header carefully.

The header ends with the first blank line. The message itself (or *body*) starts right after. The next header in the file will once again start with a `From`. `From`'s on the beginning of a line **never** exist within the body. If they do, the mailbox is considered to be corrupt.

Some mail readers store their messages in a different format. However the above format (called the *mbox* format) is the de facto standard.

9.1 Sending and reading mail

The simplest way to send mail is to use the `mail` command. Type `mail -s "hello there" <username>`. `mail` will then wait for you to type out your message. When you are finished, enter a `.` on its own on a single line. The username will be another user on your system. If no one else is on your system then send mail to `root` with `mail -s "Hello there" root` or `mail -s "Hello there" root@localhost` (if the @ is not present then local machine, `localhost`, is implied).

You can use `mail` to view your mailbox. This is a primitive utility in comparison to modern graphical mail readers but is probably the only mail reader that can handle arbitrary sized mailboxes. Sometimes you may get a mailbox that is over a gigabyte in size, and `mail` is the only way to delete messages from it. To view your mailbox, type `mail`, and then `z` to read your next window of messages, and `z-` to view the previous window. Most commands work like *command message_number*, eg `delete 14` or `reply 7` etc. The message number is the left column with an N next to it for new mail, etc.

For the state of the art in terminal based mail readers, try `mutt` and `pine`.

There are also some graphical mail readers in various stages of development. At the time I am writing this, I have been using `TkRat` for a few months, which was the best mail reader I could find.

9.2 The SMTP protocol — sending mail raw to port 25

To send mail, it is actually not necessary to have a mail client at all. The mail client just follows *SMTP* (Simple Mail Transfer Protocol), which you can type in from the keyboard.

For example, you can send mail by `telnet`*ing* to *port 25* of a machine that has an *MTA* (Mail Transfer Agent — also called the *mailer daemon*) running.

This is in fact how, so-called, *anonymous mail* or *spam mail*[1] is sent on the Internet. A mailer daemon runs in most small institutions in the world, and has the simple task of receiving mail requests and relaying them onto other mail servers. Try this for example (obviously substituting `mail.obsidian.co.za` for the name of a mail server that you normally use):

```
[root@cericon tex]# telnet mail.obsidian.co.za 25
Trying 192.168.2.1...
Connected to 192.168.2.1.
```

[1] *Spam* is a term used to indicate unsolicited email — that is junk mail that is posted in bulk to large numbers of arbitrary email address. This is considered unethical Internet practice.

```
Escape character is '^]'.
220 ra.obsidian.co.za ESMTP Sendmail 8.9.3/8.9.3; Wed, 2 Feb 2000 14:54:47 +0200
HELO cericon.obsidian.co.za
250 ra.obsidian.co.za Hello cericon.ctn.obsidian.co.za [192.168.3.9], pleased to meet you
MAIL FROM:psheer@icon.co.za
250 psheer@icon.co.za... Sender ok
RCPT TO:maynard@iafrica.com
250 maynard@iafrica.com... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
hi there
heres a short message

.
250 OAA04620 Message accepted for delivery
QUIT
221 ra.obsidian.co.za closing connection
Connection closed by foreign host.
[root@cericon tex]#
```

The above causes the message "`hi there here is a short message`" to be delivered to `maynard@iafrica.com` (the `ReCiPient`). Of course I can enter any address that I like as the sender, and it can be difficult to determine who sent the message.

Now, you may have tried this and got a rude error message. This might be because the MTA is configured **not** to relay mail except from specific trusted machines — say only those machines within that organisation. In this way anonymous email is prevented.

Chapter 10

Managing User Accounts and User Ownerships

10.1 Users and Groups

UNIX intrinsically[1] supports multiple users. Each user has a personal *home* directory `/home/<username>` in which their own files are stored, hidden from other users.

So far you may have been using the machine as the `root` user, who is the system administrator and has complete access to every file on the system. The home directory of the `root` user is `/root`. **Note that there is an ambiguity here: the *root* directory is the top most directory, known as the `/` directory. The `root` user's home directory is `/root` and is called the *home directory of `root`*.**

Other than `root`, every other user has limited access to files and directories. Always use your machine as a normal user. Login as `root` only to do system administration. This will save you from the destructive power that the `root` user has. Here we will show how to manually and automatically create new users.

Users are also divided into sets, called *groups*. A user may belong to several groups and there can be as many groups on the system as you like. Each group is defined by a list of users that are part of that set. In addition each user has a group of the same name, to which only he belongs.

10.2 File ownerships

Each file on a system is *owned* by a particular user and also *owned* by a particular group. When you do an `ls -al` you can see the user that owns the file in the third column and the group that owns the file in the fourth column (these will often be identical indicating that the file's group is a group to which only the user belongs). To change the ownership of the file simply use the `chown`, *change ownerships*, command as follows.

```
chown <user>[:<group>] <filename>
```

10.3 The password file `/etc/passwd`

The only place in the whole system where a user name is registered is in this file[2]. Once a user is added to this file, they *exist* on the system[3]. This is also known as the *password* file to administrators. View this file with `less`:

```
root:x:0:0:Paul Sheer:/root:/bin/bash
```

[1] To the innermost degree

[2] One exception to this rule is the SMB package which emulates a Windows NT shared network drive. Because of the rather odd way that Windows stores encrypted passwords, a completely separate password file is required for this package int `/etc/smbpasswd`

[3] If you might have thought that user accounts where stored in some unreachable dark corner then this should dispel this idea.

```
bin:x:1:1:bin:/bin:
daemon:x:2:2:daemon:/sbin:
adm:x:3:4:adm:/var/adm:
lp:x:4:7:lp:/var/spool/lpd:
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:
news:x:9:13:news:/var/spool/news:
uucp:x:10:14:uucp:/var/spool/uucp:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
ftp:x:14:50:FTP User:/home/ftp:
nobody:x:99:99:Nobody:/:
alias:x:501:501::/var/qmail/alias:/bin/bash
paul:x:509:510:Paul Sheer:/home/paul:/bin/bash
jack:x:511:512:Jack Robbins:/home/jack:/bin/bash
silvia:x:511:512:Silvia Smith:/home/silvia:/bin/bash
```

Above is an extract of my own password file. Each user is stored on a separate line. Many of these are not human login accounts, but are used by other programs.

Each line contains seven *fields* separated by colons. The account for `jack` looks like this:

`jack` The users login name.

`x` The users encrypted password. If this is an `x`, it indicates that it is stored in a separate file, `/etc/shadow`. This *shadow* password file is a later addition to UNIX systems that contains additional information about the user.

`511` The user's user identification number, *UID*[4].

`512` The user's group identification number, *GID*[5].

`Jack Robbins` The user's full name[6].

`/home/jack` The user's home directory. The `HOME` environment variable will be set to this when the user logs in.

`/bin/bash` The shell to start when the user logs in.

10.4 The shadow password file `/etc/shadow`

The problem with traditional `passwd` files is that they had to be *world readable*[7] in order for programs to extract information about the user: such as the users full name. This means that everyone can see the encrypted password in the second field. Anyone can copy any other user's password field and then try billions of different passwords to see if they match. If you have a hundred users on the system, there is bound to be several that chose passwords that match some word in the dictionary. The so-called *dictionary* attack will simply try all 80000 English words until a match is found. If you think you are clever to add a number in front of an easy-to-guess dictionary word, password cracking algorithms know about these as well[8]. To solve this problem the `shadow` password file was invented. The shadow password file is used only for *authentication*[9] and is not world readable — there is no information in the shadow password file that a common program will ever need — no regular user has permission see the encrypted password field. The fields are colon separated just like the `passwd` file.

Here is an example line from a `/etc/shadow` file:

```
jack:Q,Jpl.or6u2e7:10795:0:99999:7:-1:-1:134537220
```

[4] This is used by programs as short alternative to the users login name. In fact, internally, the login name is never used, only the UID.
[5] Similar applies to the GID. Groups will be discussed later
[6] Few programs ever make use of this field.
[7] Everyone on the system can read the file
[8] And about every other trick you can think of.
[9] Verifying that the user is the genuine owner of the account.

`jack` The user's login name.

`Q,Jpl.or6u2e7` The user's encrypted password known as the *hash* of the password. This is the user's 8 character password with a *one way hash function* applied to it. It is simply a mathematical algorithm applied to the password that is known to produce a unique result for each password. To demonstrate: the (rather poor) password `Loghimin` hashes to `:lZ1F.0VSRRucs:` in the shadow file. An almost identical password `loghimin` gives a completely different hash `:CavHIpD1W.cmg:`. Hence trying to guess the password from the hash can only be done by trying every possible password, and is therefore considered computationally expensive *but not impossible*. To check if an entered password matches, just apply the identical mathematical algorithm to it: if it matches then the password is correct. This is how the login command works. Sometimes you will see a `*` in place of a hashed password. This means that the account has been disabled.

`10795` Days since the January 1, 1970 that the password was last changed.

`0` Days before which password may not be changed. Usually zero. This field is not often used.

`99999` Days after which password must be changed. This is also rarely used, and will be set to 99999 by default.

`7` Days before password is to expire that user is warned of pending password expiration.

`-1` Days after password expires that account is considered inactive and disabled. `-1` is used to indicate infinity — i.e. to mean we are effectively not using this feature.

`-1` Days since January 1, 1970 when account will be disabled.

`134537220` Flag reserved for future use.

10.5 The groups file /etc/group and the groups command

On a UNIX system you may want to give a number of users the same access rights. For instance, you may have five users that should be allowed to access some privileged file, and another ten users that are allowed to run a certain program. You can *group* these users into, for example, two groups `previl` and `wproc` and then make the relevant file and directories owned by that group with, say,

```
chown root:previl /home/somefile
chown root:wproc /usr/lib/wproc
```

Permissions[10] will dictate the kind of access, but for the mean time, the file/directory must at least be *owned* by that group.

The `/etc/group` file is also colon separated. A line might look like this:

```
wproc:x:524:jack,mary,henry,arthur,sue,lester,fred,sally
```

wproc The name of the group. There should really also be a user of this name as well.

x The groups password. This field is usually set with an `x` and is not used.

524 The GID *group ID*. This must be unique in the groups file.

jack,mary,henry,arthur,sue,lester,fred,sally The list of users that belong to the group. This must be comma separated with no spaces.

The groups command

You can obviously study the `group` file to find out which groups a user belongs to[11], but when there are a lot of groups it can be tedious to scan through the entire file. The `groups` command prints out this information.

[10] explained later.
[11] That is, not "which users is a group comprised of", which is easy to see at a glance.

10.6 Manually creating a user account

The following steps will create a user account:

`/etc/passwd` **entry** To create an entry in this file, simply edit it and copy an existing line[12]. Always add users from the bottom and try to preserve the "pattern" of the file — i.e. if you see numbers increasing, make yours fit in; if you are adding a normal user, add it after the existing lines of normal users. Each user must have a unique UID and should usually have a unique GID. So if you are adding a line to the end of the file, make your new UID and GID the same as the last line but incremented by one.

`/etc/shadow` **entry** Create a new shadow password entry. At this stage you do not know what the hash is, so just make it a `*`. You can set the password with the `passwd` command later.

`/etc/group` **entry** Create a new group entry for the user's group. Make sure the number in the group entry matches that in the `passwd` file.

`/etc/skel` This directory contains a template home directory for the user. Copy the entire directory and all its contents into `/home` directory, renaming it to the name of the user. In the case of our `jack` example, you should have a directory `/home/jack`.

Home directory ownerships You need to now change the ownerships of the home directory to match the user. The command `chown -R jack:jack /home/jack` will accomplish this.

Setting the password Use `passwd <username>` to set the users password.

10.7 Automatically creating a user account — `useradd` and `groupadd`

The above process is tedious. Two commands that perform all these updates automatically are `useradd`, `userdel` and `usermod`. The man pages will explain the use of these commands in detail. Note that different flavours of UNIX have different commands to do this. Some may even have graphical programs or web interfaces to assist in creating users.

In addition, there are the commands `groupadd`, `groupdel` and `groupmod` which do the same with respect to groups.

10.8 User logins

The `login` command

A user most often gains access to the system through the `login` program. This looks up the UID and GID from the `passwd` and `group` file, and authenticates the user.

The following is quoted from the `login` man page:

> **login** is used when signing onto a system. It can also be used to switch from one user to another at any time (most modern shells have support for this feature built into them, however).
>
> If an argument is not given, **login** prompts for the username.
>
> If the user is *not* root, and if */etc/nologin* exists, the contents of this file are printed to the screen, and the login is terminated. This is typically used to prevent logins when the system is being taken down.
>
> If special access restrictions are specified for the user in */etc/usertty*, these must be met, or the log in attempt will be denied and a **syslog**[13] message will be generated. See the section on "Special Access Restrictions".

[12] When editing configuration files, never write out a line from scratch if it has some kind of special format. Always copy an existing entry that has proven itself to be correct, and then edit in the appropriate changes. This will prevent you from making errors

[13] System error log program — `syslog` writes all system messages are to the file `/var/log/messages]`

If the user is root, then the login must be occuring on a tty listed in */etc/securetty*[14]. Failures will be logged with the **syslog** facility.

After these conditions are checked, the password will be requested and checks (if a password is required for this username). Ten attempts are allowed before **login** dies, but after the first three, the response starts to get very slow. Login failures are reported via the **syslog** facility. This facility is also used to report any successful root logins.

If the file *.hushlogin* exists, then a "quiet" login is performed (this disables the checking of the checking of mail and the printing of the last login time and message of the day). Otherwise, if */var/log/lastlog* exists, the last login time is printed (and the current login is recorded).

Random administrative things, such as setting the UID and GID of the tty are performed. The TERM environment variable is preserved, if it exists (other environment variables are preserved if the **-p** option is used). Then the HOME, PATH, SHELL, TERM, MAIL, and LOGNAME environment variables are set. PATH defaults to */usr/local/bin:/bin:/usr/bin:.*[16] for normal users, and to */sbin:/bin:/usr/sbin:/usr/bin* for root. Last, if this is not a "quiet" login, the message of the day is printed and the file with the user's name in */usr/spool/mail* will be checked, and a message printed if it has non-zero length.

The user's shell is then started. If no shell is specified for the user in **/etc/passwd**, then **/bin/sh** is used. If there is no directory specified in */etc/passwd*, then / is used (the home directory is checked for the *.hushlogin* file described above).

The *set user*, su command

To temporarily become another user, you can use the su program:

```
su jack
```

This will prompt you for a password unless you are the root user to start off with. This does nothing more than change the current user to have the access rights of jack. Most environment variables will remain the same. The HOME, LOGNAME and USER environment variables will be set to jack, but all other environment variables will be inherited. su is therefore not the same as a normal login.

To use su to give you the equivalent of a login, do

```
su - jack
```

This will cause all initialisation scripts that are normally run when the user logs in to be executed[17]. Hence after running su with the - option, you are as though you had logged in with the login command.

The who, w and users commands to see who is logged in

who and w gives list of users logged into the system and how much CPU they are using etc. who --help gives:

```
Usage: who [OPTION]... [ FILE | ARG1 ARG2 ]

  -H, --heading      print line of column headings
  -i, -u, --idle     add user idle time as HOURS:MINUTES, . or old
  -m                 only hostname and user associated with stdin
  -q, --count        all login names and number of users logged on
  -s                 (ignored)
  -T, -w, --mesg     add user's message status as +, - or ?
      --message      same as -T
      --writable     same as -T
```

[14] If this file is not present, then root logins will be allowed from anywhere. It is worth deleting this file if your machine is protected by a firewall and you would like to easily login from another machine on your LAN[15]. If /etc/securetty is present, then logins are only allowed from the terminals it lists.

[16] Note that the . — the current directory — is listed in the PATH. This is only the default PATH however.

[17] What actually happens is that the subsequent shell is started with a - in front of the zero'th argument. This makes the shell read the user's personal profile. The login command also does this.

```
      --help         display this help and exit
      --version      output version information and exit

If FILE is not specified, use /var/run/utmp.  /var/log/wtmp as FILE is common.
If ARG1 ARG2 given, -m presumed: 'am i' or 'mom likes' are usual.
```

A little more information can be gathered from the `info` pages for this command. The idle time indicates how long since the user has last pressed a key. Most often, one just types `who -Hiw`.

`w` is similar. Its man page says:

> `w` displays information about the users currently on the machine, and their processes. The header shows, in this order, the current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.
>
> The following entries are displayed for each user: login name, the tty name, the remote host, login time, idle time, JCPU, PCPU, and the command line of their current process.
>
> The JCPU time is the time used by all processes attached to the tty. It does not include past background jobs, but does include currently running background jobs.
>
> The PCPU time is the time used by the current process, named in the "what" field.

Finally, from a shell script the `users` command is useful for just seeing who is logged in. You can use in a shell script, for example:

```
for user in `users` ; do
    <etc>
```

The `id` command and *effective* UID

`id` prints your *real* and *effective* UID and GID. A user will normally have a UID and a GID but may also have an effective UID and GID as well. The real UID and GID are what a process will generally think you are logged in as. The effective UID and GID are the actual access permissions that you have when trying to read, write and execute files. These will be discussed in more detail later.

Chapter 11

Using Internet Services

This chapter should make you aware of the various methods for transferring files and data over the Internet, and remotely accessing UNIX machines.

11.1 `telnet` and `rlogin`

`telnet` is a program for talking to a UNIX network service. It is most often used to do a remote login. Try

```
telnet <remote_machine>
telnet localhost
```

to login to your remote machine. It needn't matter if there is no physical network, network services always work regardless, because the machine always has an internal link to itself.

`rlogin` is like a minimal version of `telnet` that allows login access only. You can type

```
rlogin -l <username> <remote_machine>
rlogin -l jack localhost
```

if the system is configured to support remote logins.

11.2 FTP

FTP stands for *File Transfer Protocol*. If FTP is set up on your local machine, then other machines can download files. Type

```
ftp metalab.unc.edu
```

or

```
ncftp metalab.unc.edu
```

`ftp` was the tradition command-line UNIX FTP *client*[1], while *ncftp* is a more powerful client that will not always be installed.

You will now be inside an FTP *session*. You will be asked for a login name and a password. The site `metalab.unc.edu` is one that allows *anonymous* logins. This means that you can type `anonymous` as your username, and then anything you like as a password. You will notice that it will ask you for an email address as your password. Any sequence of letters with a `@` symbol will suffice, but you should put your actual email address out of politeness.

[1] "*client*" always indicates the user program accessing some remote service.

The FTP session is like a reduced shell. You can type `cd`, `ls` and `ls -al` to view file lists. `help` brings up a list of commands and you can also type `help <command>` to get help on a specific command. You can download a file using the `get <filename>` command, but before you do this, you must set the *transfer type* to *binary*. The *transfer type* indicates whether or not new-line characters will be translated to DOS format or not. Typing *ascii* turns this on, while *binary* turns it off. You may also want to enter *hash* which will print a `#` for every 1024 bytes of download. This is useful to watch the progress of a file. Goto a directory that has a `README` file in it and enter,

```
get README
```

The file will be downloaded into your current directory.

You can also `cd` to the `/incoming` directory and upload files. Try,

```
put README
```

to upload the file that you have just downloaded. Most FTP sites have an `/incoming` directory which is flushed periodically.

FTP allows far more than just uploading of files, although the administrator has the option to restrict access to any further features. You can create directories, change ownerships and do almost anything you can on a local file system.

If you have several machines on a LAN, all should have FTP enabled to allow users to easily copy files between machines. configuring the FTP server will be dealt with later.

11.3 finger

`finger` is a service for telling who is logged in on a remote system. Try `finger @<hostname>` to see who is logged in on `<hostname>`. The *finger* service will often be disabled on machines for security reasons.

11.4 Sending files by email

uuencode and uudecode

Mail is becoming used more and more for transferring files between machines. It is bad practice to send mail messages over 64 kilobytes over the Internet because it tends to excessively load mail servers. Any file larger than 64 kilobytes should be uploaded by FTP onto some common FTP server. Most small images are smaller than this size, hence sending a small JPEG[2] image is considered acceptable.

To send files by mail if you have to is best accomplished using `uuencode`. This utility takes binary files and packs them into a format that mail servers can handle. If you send a mail message containing arbitrary binary data, it will more than likely be corrupted in the way, because mail agents are only designed to handle a limited range of characters. `uuencode` takes a binary file and represents it in allowable characters albeit taking up slightly more space.

Here is a neat trick to pack up a directory and send it to someone by mail.

```
tar -czf - <mydir> | uuencode <mydir>.tar.gz \
    | mail -s "Here are some files" <user>@<machine>
```

To unpack a `uuencode`d file, use the `uudecode` command:

```
uudecode <myfile>.uu
```

[2] A common internet image file format. These are especially compressed and are usually under 100 kilobytes for a typical screen sized photograph.

MIME encapsulation

Most mail readers have the ability to *attach* files to mail messages and read these attachments. The way they do this is not with `uuencode` but in a special format known as *MIME encapsulation*. MIME is way of representing multiple files inside a single mail message. The way binary data is handled is similar to `uuencode`, but in a format known as *base64*.

If needed, there are two useful command-line utilities in the same vein as `uuencode` that can create and extract MIME messages. These are `mpack` and `munpack`.

Chapter 12

Linux resources

Very often it is not even necessary to connect to the Internet to find the information you need. Chapter 15 contains a description of most of the documentation on a LINUX distribution.

It is however essential to get the most up to date information where security and hardware driver support is concerned. It is also fun and worthwhile to interact with LINUX users from around the globe. The rapid development of free software could mean that you may miss out on important new features that could streamline IT services. Hence reviewing web news, reading newsgroups and subscribing to mailing lists are essential parts of a system administrators role.

12.1 FTP sites and the sunsite mirror

The metalab.unc.edu FTP site is considered the primary site for free software the world over. It is mirrored in almost every country that has a significant IT infrastructure.

Our local South Africa mirror's are ftp.is.co.za in the directory /linux/sunsite, and also somewhere on the site ftp.sdn.co.za.

It is advisable to browse around these ftp sites. In particular you should try to find the locations of:

- The directory where all sources for official GNU packages are stored. This would be a mirror of the Free Software Foundation's FTP archives. These are packages that were commissioned by the FSF, and not merely released under the GPL. The FSF will distribute them in source form (.tar.gz) for inclusion into various distributions. They will of course compile and work under any UNIX.
- The mirror of the metalab. This is known as the sunsite mirror because it used to be called metalab.unc.edu. It contains enumerable UNIX packages in source and binary form, categorised in a directory tree. For instance, mail clients have their own directory with many mail packages inside. metalab is the place where a new developer can host some new software that they have produced. There are instructions on the FTP site to upload software and to request it to be placed into a directory.
- The kernel sources. This is a mirror of the kernel archives where Linus and other maintainers upload new *stable*[1] and *beta*[2] kernel versions and kernel patches.
- The various distributions. RedHat, Debian and possibly other popular distributions will be mirrored.

This list is by no means exhaustive. Depending on the willingness of the site maintainer, there may be mirrors to far more sites from around the world.

The FTP site is how you will download free software. Often, maintainers will host their software on a a web site, but every popular package will almost always have an FTP site where versions are persistently stored. An example is lava.obsidian.co,za in the directory /pub/linux/cooledit where the author's own **Cooledit** package is distributed — the layout is typical of an FTP site.

[1] Meaning that the software is well tested and free of serious bugs.
[2] Meaning that the software is in its development stages.

12.2 HTTP — web sites

Most users should already be familiar with using a web browser. You should also become familiar with the concept of a *web search*. This is when you point your web browser to a popular search engine like http://infoseek.go.com/, http://www.altavista.com/ or http://www.yahoo.com/ and search for a particular key word. Searching is a bit of a black art with the billions of web pages out there. Always consult the search engine's advanced search options to see how you can do more complex searches than just plain word searches.

The web sites in the FAQ (excluding the list of known distributions) should all be consulted to get a overview on some of the primary sites of interest to LINUX users.

Especially important is that you keep up to do with the latest LINUX news. I find the *Linux Weekly News* http://lwn.net/ excellent for this. Also, The famous (and infamous) *SlashDot* http://slashdot.org/ web site gives daily updates about "stuff that matters" and therefore contains a lot about free software.

Fresh Meat http://frshmeat.net/ is a web site devoted to new software releases. You will find new or updated packages uploaded every few hours or so.

Linux Planet http://www.linuxplanet.com/ seems to be a new (?) web site that I just found while writing this. It looks like it contains lots of tutorial information on LINUX.

Realistically though, a new LINUXweb site is created every week; almost anything prepended or append to "linux" is probably a web site already.

12.3 Mailing lists

A mailing list is a special address that when posted to, automatically sends email a long list of other addresses. One usually subscribes to a mailing list by sending some especially formatted email, or requesting a subscription from the mailing list manager.

Once you have subscribed to a list, any email you post to the list will be sent to every other subscriber, and every other subscribers posts to the list will be sent to you.

There are mostly three types of mailing lists. Those over the *majordomo* type, those of the *listserv* type, and those of the **-request* type.

Majorodom and Listserv

To subscribe to the *majordomo* type variety send mail message to majordomo@<machine> with no subject and a one line message:

```
subscribe <mailing-list-name>
```

This will subscribe you to the mailing list <mailing-list-name>@<machine>, where messages are posted to.

Do the same for *listserv* type lists, by sending the same message to listserv@<machine>.

For instance, if you are an administrator for any machine that is exposed to the Internet you should get on bugtraq. Send an email

```
subscribe bugtraq
```

to listserv@netspace.org, and become one of the tens of thousands of users that read and report security problems about LINUX.

To *unsubscribe* to a list is just as simple, send an email message,

```
unsubscribe <mailing-list-name>
```

Never send subscribe or unsubscribe messages to the mailing list itself. Send subscribe or unsubscribe messages only to to the address majordomo@<machine> or listserv@<machine>.

*-request

These can be subscribed to by sending an empty email message to <mailing-list-name>-request@<machine> with the word subscribe as the subject. The same email with the word unsubscribe will remove you from the list.

Once again, never send subscribe or unsubscribe messages to the mailing list itself..

12.4 Newsgroups

A newsgroup is a notice board that everyone in the world can see. There are tens of thousands of newsgroups and each group is unique in the world.

The client software you will use to read a newsgroup is called a *news reader*. rtin is a popular text mode reader, while netscape is graphical.

Newsgroups are named like Internet hosts. One you might be interested in is comp.os.linux.announce. The comp is the broadest subject description for *computers*, os stands for *operating systems*, etc. There are many other linux newsgroups devoted to various LINUX issues.

Newsgroups servers are big hungry beasts. They form a tree like structure on the Internet. When you send mail to a newsgroups it takes about a day or so for the mail you sent to propagate to every other server in the world. Likewise you can see a list of all the messages posted to each newsgroup by anyone anywhere.

Whats the difference between a newsgroup and a mailing list? The advantage of a newsgroup is that you don't have to download the messages your are not interested in. If you are on a mailing list, you get all the mail sent to the list. With a newsgroup you can look at the message list and retrieve only the messages you are interested in.

Why not just put the mailing list on a web page? If you did, then everyone in the world will have to go over international links to get to the web page. It would load the server in proportion to the number of subscribers. This is exactly what SlashDot is. However your newsgroup server is local, hence you retrieve mail over a faster link and save Internet traffic.

Chapter 13

Permission and Modification Times

13.1 Permissions

Every file and directory on a UNIX system, besides being owned by a user and a group, has access *flags*[1] dictating what kind of access that user and group has to the file.

Doing an `ls -ald /bin/cp /etc/passwd /tmp` will give you a listing:

```
-rwxr-xr-x    1 root      root         28628 Mar 24   1999 /bin/cp
-rw-r--r--    1 root      root          1151 Jul 23 22:42 /etc/passwd
drwxrwxrwt    5 root      root          4096 Sep 25 15:23 /tmp
```

In the left most column are these flags, which give a complete description of the access rights to the file.

The furthest flag to the left is, so far, either `-` or `d` indicating an ordinary file or directory. The remaining nine have a `-` to indicate an unset value or one of several possible characters. Table 13.1 gives a complete description of file system permissions.

The `chmod` command is used to change the permissions of a file. It usually used like:

```
chmod [-R] [u|g|o|a][+|-][r|w|x|s|t] <file> [<file>] ...
```

For example

```
chmod u+x myfile
```

adds execute permissions for the user of myfile. And,

```
chmod a-rx myfile
```

removes *read* and *execute* permissions for *all* — i.e. user, group and other.

The `-R` options once again means *recursive*, diving into subdirectories as usual.

Permission bits are often represented in their binary form, especially when programming. It is convenient to show the `rwxrwxrwx` set in *octal*, where each digit fits conveniently into three bits. Files on the system are usually created with *mode* `0644`, meaning `rw-r--r--`. You can set permissions explicitly with an octal number:

```
chmod 0755 myfile
```

Gives `myfile` the permissions `rwxr-xr-x`.

In the table you can see `s`, the *setuid* or *setgid* bit. If it is used without execute permissions then it has no meaning and is written capitalised as an `S`. This bit effectively colourises a `x` into an `s`, hence you should read

[1] A switch that can either be on or off

	Possible chars, - for unset	Effect for directories	Effect for files
User, u	r	User can read the contents of the directory.	User can read the file.
	w	With x or s, user can create and remove files in the directory.	User can write to the file.
	x s S	User can access the contents of the files in a directory for x or s. S has no effect.	User can execute the file for x or s. s, known as the *setuid* bit, means to set the user owner of the subsequent process to that of the file. S has no effect.
Group, g	r	Group can read the contents of the directory.	Group can read the file.
	w	With x or s, group can create and remove files in the directory.	Group can write to the file.
	x s S	Group can access the contents of the files in a directory for x. For s force all files in this directory to the same group as the directory. S has no effect.	Group can execute the file for x or s. s, known as the *setgid* bit, means to set the group owner of the subsequent process to that of the file. S has no effect.
Other, o	r	Everyone can read the contents of the directory.	Everyone can read the file.
	w	With x or t, everyone can create and remove files in the directory.	Everyone can write to the file.
	x t T	Everyone can access the contents of the files in a directory for x and t. t, is known as the *sticky* bit, and prevents users from removing files that they do not own, hence they are free to append to the directory, but not remove other users' files. T has no effect.	Group can execute the file for x or t. For t save the processes text image to the swap device so that future loads will be faster (I don't know if this has an effect on LINUX). T has no effect.

Table 13.1: File and directory permissions

and s as *execute* **with** the setuid or setgid bit set. t is known as the sticky bit. It also has no meaning if there are no execute permissions and is written as a capital T.

The leading 0 can in be ignored, but is preferred in order to be explicit. It *can* take on a value representing the three bits, *setuid* (4), *setgid* (2) and *sticky* (1). Hence a value of 5764 is 101 111 110 100 in binary and gives -rwsrw-r-T.

13.2 Modification times and the stat command

In addition to permissions, each file has three integers associated with it that represent in seconds, the last time the file was accessed (read), when it was last modified, and when it was created. These are known as the *atime*, *mtime* and *ctime* of a file respectively.

To get a complete listing of the file's permissions, use the stat command. Here is the result of stat /etc:

```
  File: "/etc"
  Size: 4096          Filetype: Directory
  Mode: (0755/drwxr-xr-x)          Uid: (    0/    root)  Gid: (    0/    root)
Device:  3,1    Inode: 14057      Links: 41
Access: Sat Sep 25 04:09:08 1999(00000.15:02:23)
Modify: Fri Sep 24 20:55:14 1999(00000.22:16:17)
Change: Fri Sep 24 20:55:14 1999(00000.22:16:17)
```

The Size: quoted here is the actual amount of disk space used in order to store the directory **listing**, and is the same as reported by ls. In this case it is probably four disk blocks of 1024 bytes each. The size of a directory

as quoted here does **not** mean the sum of all files contained under it.

Chapter 14

Symbolic and Hard Links

14.1 Soft links

Very often, a file is required to be in two different directories at the same time. Think for example of a configuration file that is required by two different software packages that are looking for the file in different directories. The file could simple be copied, but this would create an administrative nightmare to have to replicate changes in more than one place. The way two files can have the same data is with links.

Try

```
touch myfile
ln -s myfile myfile2
ls -al
cat > myfile
a
few
lines
of
text
^C
cat myfile
cat myfile2
```

You will notice that the `ls -al` listing has the letter `l` on the far left next to `myfile2` while the usual `-` next to `myfile`. This indicates that the file is a *soft* link (also known as a *symbolic* link or *symlink*) to some other file.

A *symbolic link* contains no data of its own, only a reference to another file. It can even contain a reference to a directory. In either case, programs operating on the link will actually see the file or directory it points to.

Try

```
mkdir mydir
ln -s mydir mydir2
ls -al .
touch ./mydir/file1
touch ./mydir2/file2
ls -al ./mydir
ls -al ./mydir2
```

The directory `mydir2` is a symbolic link to `mydir2` and appears as though it is a replica of the original. Once again the directory `mydir2` does not consume additional disk space — a program that reads from the link is unaware that it is seeing into a different directory.

Symbolic links can also be copied and retain their value:

```
cp mydir2 /
ls -al /
```

```
cd /mydir2
```

You have now copied the link to the root directory. However the link points to a relative path `mydir` in the same directory as the link. Since there is no `mydir` here, an error is raised.

Try

```
rm -f mydir2 /mydir2
ln -s `pwd`/mydir mydir2
ls -al
```

Now you will see `mydir2` has an absolute path. You can try

```
cp mydir2 /
ls -al /
cd /mydir2
```

and notice that it does now work.

One of the common uses of symbolic links is to make *mount*ed (see Section 18.4) file systems accessible from a different directory. For instance, you may have a large directory that has to be split over several physical disks. For clarity, you can mount the disks as `/disk1`, `/disk2` etc.and then link the various sub-directories in a way that makes efficient use of the space you have.

Another example is the linking of `/dev/cdrom` to, say, `/dev/hdc` so that programs accessing the device (see Chapter 17.1) file `/dev/cdrom`, actually access the correct IDE drive.

14.2 Hard links

UNIX allows the data of a file to have more than one name in separate places in the same file system. Such a file with more than one name for the same data is called a *hard linked* file and is very similar to a symbolic link. Try

```
touch mydata
ln mydata mydata2
ls -al
```

The files `mydata2` and `mydata2` are indistinguishable. They share the same data, and have a `2` in second column of the `ls -al` listing. This means that they are hard linked **twice** (that there are two names for this file).

The reason why hard links are sometimes used in preference to symbolic links is that some programs are not fooled by a symbolic link: if you, say, have a script that uses `cp` to copy a file, it will copy the symbolic link instead of the file it points to[1]. A hard link however will always be seen as a real file.

Hard links however cannot be made between files on different file-systems. They also cannot be made between directories.

[1] `cp` actually has an option to override this behaviour

Chapter 15

Pre-installed Documentation

This chapter describes where to find documentation on a common LINUX distribution. At the moment a RedHat distribution is assumed, but this is equally applicable to other distributions, although the exact locations might be different.

For many proprietary operating systems, the definitive reference for their operating system are printed texts. For LINUX, much of documentation is written by the authors themselves and is included with the source code. A typical LINUX distribution will package this along with the compiled binaries. Common distributions come with **hundreds of megabytes** of printable, hyper-linked, and plane text documentation. There is often no need to go the the World Web Wide unless something is outdated.

If you have not already tried this, run

```
ls -ld /usr/*/doc /usr/*/*/doc /usr/share/*/*/doc \
       /opt/*/doc /opt/*/*/doc
```

This is a somewhat unreliable way to search for potential documentation directories.

Kernal documentation: /usr/src/linux/Documentation/

This contains information on all hardware drivers except graphics cards. The kernel has built in drivers for networking cards, SCSI controllers, sound cards and so on. Hence if you need to find out if one of these is supported, this is the first place to look.

X Window System graphics hardware support: /usr/X11R6/lib/X11/doc/

X installs in a separate tree, In here you will find documentation on all of the graphics hardware supported by X, how to configure X, tweak video modes, cope with incompatible graphics cards, and so on.

TEX and Metafont reference: /usr/share/texmf/doc/

This is an enormous and comprehensive (and possibly exhaustive) reference to the TEX typesetting language and the Metafont font generation package.

LATEX HTML documentation: /usr/share/texmf/tex/latex/

This is a complete reference to the LATEX typesetting language. (This book itself was typeset using LATEX.)

Frequently Asked Questions: /usr/doc/FAQ

This contains some beginners documentation.

Howto's: /usr/doc/HOWTO

This is an excellent source of laymen tutorials for setting up almost any kind of service you can imagine. It is worth listing the contents here to emphasise diversity of topics covered:

```
3Dfx-HOWTO                  Danish-HOWTO                Intranet-Server-HOWTO       PPP-HOWTO                   Software-Release-Practice-HOWTO
AX25-HOWTO                  Distribution-HOWTO          Italian-HOWTO               PalmOS-HOWTO                Sound-HOWTO
Access-HOWTO                ELF-HOWTO                   Java-CGI-HOWTO              Parallel-Processing-HOWTO   Sound-Playing-HOWTO
Alpha-HOWTO                 Emacspeak-HOWTO             Kernel-HOWTO                Pilot-HOWTO                 Spanish-HOWTO
Assembly-HOWTO              Esperanto-HOWTO             Keyboard-and-Console-HOWTO  Plug-and-Play-HOWTO         TeTeX-HOWTO
Bash-Prompt-HOWTO           Ethernet-HOWTO              KickStart-HOWTO             Polish-HOWTO                Text-Terminal-HOWTO
Benchmarking-HOWTO          Finnish-HOWTO               LinuxDoc+Emacs+Ispell-HOWTO Portuguese-HOWTO            Thai-HOWTO
Beowulf-HOWTO               Firewall-HOWTO              META-FAQ                    PostgreSQL-HOWTO            Tips-HOWTO
BootPrompt-HOWTO            French-HOWTO                MGR-HOWTO                   Printing-HOWTO              UMSDOS-HOWTO
Bootdisk-HOWTO              Ftape-HOWTO                 MILO-HOWTO                  Printing-Usage-HOWTO        UPS-HOWTO
Busmouse-HOWTO              GCC-HOWTO                   MIPS-HOWTO                  Quake-HOWTO                 UUCP-HOWTO
CD-Writing-HOWTO            German-HOWTO                Mail-HOWTO                  README                      \unix{}-Internet-Fundamentals-HOWTO
CDROM-HOWTO                 Glibc2-HOWTO                Modem-HOWTO                 RPM-HOWTO                   User-Group-HOWTO
COPYRIGHT                   HAM-HOWTO                   Multi-Disk-HOWTO            Reading-List-HOWTO          VAR-HOWTO
Chinese-HOWTO               Hardware-HOWTO              Multicast-HOWTO             Root-RAID-HOWTO             VME-HOWTO
Commercial-HOWTO            Hebrew-HOWTO                NET-3-HOWTO                 SCSI-Programming-HOWTO      VMS-to-Linux-HOWTO
Config-HOWTO                INDEX.html                  NFS-HOWTO                   SMB-HOWTO                   Virtual-Services-HOWTO
Consultants-HOWTO           INFO-SHEET                  NIS-HOWTO                   SRM-HOWTO                   WWW-HOWTO
Cyrillic-HOWTO              IPCHAINS-HOWTO              Networking-Overview-HOWTO   Security-HOWTO              WWW-mSQL-HOWTO
DNS-HOWTO                   IPX-HOWTO                   Optical-Disk-HOWTO          Serial-HOWTO                XFree86-HOWTO
DOS-Win-to-Linux-HOWTO      IR-HOWTO                    Oracle-HOWTO                Serial-Programming-HOWTO    XFree86-Video-Timings-HOWTO
DOS-to-Linux-HOWTO          ISP-Hookup-HOWTO            PCI-HOWTO                   Shadow-Password-HOWTO       XWindow-User-HOWTO
DOSEMU-HOWTO                Installation-HOWTO          PCMCIA-HOWTO                Slovenian-HOWTO
```

Mini Howto's: /usr/doc/HOWTO/mini

These are smaller quickstart tutorials in the same vein:

```
3-Button-Mouse              Coffee                      IP-Masquerade               Mail2News                   Proxy-ARP-Subnet            StarOffice
ADSL                        Colour-ls                   IP-Subnetworking            Man-Page                    Public-Web-Browser          Term-Firewall
ADSM-Backup                 Cyrus-IMAP                  ISP-Connectivity            Modules                     Qmail+MH                    TkRat
AI-Alife                    DHCP                        Install-From-ZIP            Multiboot-with-LILO         Quota                       Token-Ring
Advocacy                    DHCPcd                      Kerneld                     NCD-X-Terminal              RCS                         Ultra-DMA
Alsa-sound                  DPT-Hardware-RAID           LBX                         NFS-Root                    README                      Update
Apache+SSL+PHP+fp           Diald                       LILO                        NFS-Root-Client             RPM+Slackware               Upgrade
Automount                   Diskless                    Large-Disk                  Netrom-Node                 RedHat-CD                   VAIO+Linux
Backup-With-MSDOS           Ext2fs-Undeletion           Leased-Line                 Netscape+Proxy              Remote-Boot                 VPN
Battery-Powered             Fax-Server                  Linux+DOS+Win95+OS2         Netstation                  Remote-X-Apps               Vesafb
Boca                        Firewall-Piercing           Linux+FreeBSD               News-Leafsite               SLIP-PPP-Emulator           Visual-Bell
BogoMips                    GIS-GRASS                   Linux+FreeBSD-mini-HOWTO    Offline-Mailing             Secure-POP+SSH              Windows-Modem-Sharing
Bridge                      GTEK-BBS-550                Linux+NT-Loader             PLIP                        Sendmail+UUCP               WordPerfect
Bridge+Firewall             Hard-Disk-Upgrade           Linux+Win95                 Partition                   Sendmail-Address-Rewrite    X-Big-Cursor
Bzip2                       INDEX                       Loadlin+Win95               Partition-Rescue            Small-Memory                XFree86-XInside
Cable-Modem                 INDEX.html                  Loopback-Root-FS            Path                        Software-Building           Xterm-Title
Cipe+Masq                   IO-Port-Programming         Mac-Terminal                Pre-Installation-Checklist  Software-RAID               ZIP-Drive
Clock                       IP-Alias                    Mail-Queue                  Process-Accounting          Soundblaster-AWE            ZIP-Install
```

Linux Documentation Project: /usr/doc/LDP

These are several online books in HTML format, such as the *System Administrators Guide, SAG*, the *Network Administrators Guide, NAG*, the *Linux Programmers Guide, LPG*.

Web documentation: /home/httpd/html

Some packages may install documentation here so that it goes online automatically if your web server is running.

Apache Reference: /home/httpd/html/manual

Apache keeps this reference material online, so that it is the default web page shown when you install Apache for the first time. Apache is the most popular web server.

Individual package documentation

All packages installed on the system have their own individual documentation directory. A package `foo` will most probably have a documentation directory `/usr/doc/foo`. This most often contains documentation released with the sources of the package, such as release information, feature news, example code, FAQ's that are not part of the FAQ package, etc. If you have a particular interest in a package, you should always scan its directory in `/usr/doc` or, better still, download its source distribution.

These are the `/usr/doc` directories that contained more than a trivial amount of documentation for that package. In some cases, the package had complete references. (For example, the complete Python references were contained nowhere else.)

```
BitchX                      expect-5.24          guile-1.3            lout                 pmake-2.1.33                 svgalib-1.3.1
ImageMagick-4.2.2           ext2ed-0.1           gv-3.5.8             lsof-4.42            python-docs-1.5.1            taper-6.9
SVGATextMode-1.8            f2c-19970805         ical-2.2             lynx-2.8.1           rhl-alpha-install-addend-en-6.0 texinfo-3.12f
SoundStudio-0.9.1           fetchmail-5.0.0      icewm-0.9.33         mars-nwe-0.99pl15    rhl-install-guide-en-6.0     tin-1.4_990216
TkStep-8.0.4                fileutils-4.0        inn-2.2              mgetty-1.1.14        rpm-3.0                      ucd-snmp-3.6.1
abuse-1.10-5                freetype-1.2         ipchains-1.3.8       mod_php-2.0.1        rxvt-2.6.PRE2                uucp-1.06.1
am-utils-6.0                gated-3.5.10         isapnptools-1.18     mod_php3-3.0.7       samba-2.0.3                  vim-common-5.3
bind-8.2                    gawk-3.0.3           kaffe-1.0.b4         mtools-3.9.1         sendmail                     wu-ftpd-2.4.2vr17
blt-2.4g                    gd-1.3               kernel-pcmcia-cs-2.2.5 ncurses-devel-4.2  sgml-tools                   xanim-27070
bzip2-0.9.0c                ghostscript-5.10     lesstif-0.88.1       netscape-common-4.51 shadow-utils-980403          xbanner-1.31
console-tools-19990302      gimp-1.0.1           lesstif-devel-0.88.1 p2c-1.22             slang-devel-1.2.2            xlispstat-3.52.9
cooledit-3.11.6             gimp-manual-1.0.0    libtiff-devel-3.4    pam-0.66             slrn-0.9.5.4                 xntp3-5.93
cvs-1.10.5                  glib-1.2.1           libtool-1.2f         pgp-2.6.3i-1         spice-2g6                    xpm-devel-3.4j
docbook-stylesheets-1.39    glibc-2.1.1          libungif-devel-4.1.0 pine-4.10            squid-2.2.STABLE1            xv-3.10a
dosemu-0.99.10              gtk+-1.2.1           lilo-0.21            pinfocom-3.0         ssh-1.2.26                   zsh-3.0.5
```

Manual Pages — man: /usr/man/

Manual pages were discussed in Section 4.7. There may be other directory trees that contain man pages — on some other UNIX's man pages are littered everywhere.

To convert a man page to PostScript (for printing or viewing), use for example (for the cp command),

```
groff -Tps -mandoc /usr/man/man1/cp.1 > cp.ps ; gv cp.ps
```

info pages: /usr/info/

Info pages were discussed in Section 4.8.

Chapter 16

Overview of the Unix Directory Layout

Packages

LINUX systems are divided into hundreds of small *packages* each performing some logical group of operations. On LINUX, many small self-contained packages inter-operate to give greater functionality than would large self-contained pieces of software. There is also no clear distinction between what is part of the operating system and what is an application — everything is just a package.

A software package on a RedHat type system is distributed in a single *RedHat Package Manager* (*RPM*) file that has a `.rpm` extension. On a *Debian* distribution, the equivalent is a `.deb` package file, and on the *Slackware* distribution there are Slackware `.tgz` files.

Each package will unpack to many files which are placed all over the system. Packages generally do not create major directories but unpack files to existing directories.

Note that on a newly installed system there are practically no files anywhere that do not belong to some kind of package.

Unix Directory Superstructure

The root directory on a UNIX system typically looks like:

```
drwxr-xr-x    2 root     root         2048 Aug 25 14:04 bin
drwxr-xr-x    2 root     root         1024 Sep 16 10:36 boot
drwxr-xr-x    7 root     root        35840 Aug 26 17:08 dev
drwxr-xr-x   41 root     root         4096 Sep 24 20:55 etc
drwxr-xr-x   24 root     root         1024 Sep 27 11:01 home
drwxr-xr-x    4 root     root         3072 May 19 10:05 lib
drwxr-xr-x    2 root     root        12288 Dec 15  1998 lost+found
drwxr-xr-x    7 root     root         1024 Jun  7 11:47 mnt
dr-xr-xr-x   80 root     root            0 Sep 16 10:36 proc
drwxr-xr-x    3 root     root         3072 Sep 23 23:41 sbin
drwxrwxrwt    5 root     root         4096 Sep 28 18:12 tmp
drwxr-xr-x   25 root     root         1024 May 29 10:23 usr
```

The `usr` directory typically looks like:

```
drwxr-xr-x    9 root     root         1024 May 15 11:49 X11R6
drwxr-xr-x    6 root     root        27648 Sep 28 17:18 bin
drwxr-xr-x    2 root     root         1024 May 13 16:46 dict
drwxr-xr-x  261 root     root         7168 Sep 26 10:55 doc
drwxr-xr-x    7 root     root         1024 Sep  3 08:07 etc
drwxr-xr-x    2 root     root         2048 May 15 10:02 games
drwxr-xr-x    4 root     root         1024 Mar 21  1999 i386-redhat-linux
drwxr-xr-x   36 root     root         7168 Sep 12 17:06 include
drwxr-xr-x    2 root     root         9216 Sep  7 09:05 info
```

```
drwxr-xr-x   79 root      root         12288 Sep 28 17:17 lib
drwxr-xr-x    3 root      root          1024 May 13 16:21 libexec
drwxr-xr-x   15 root      root          1024 May 13 16:35 man
drwxr-xr-x    2 root      root          4096 May 15 10:02 sbin
drwxr-xr-x   39 root      root          1024 Sep 12 17:07 share
drwxr-xr-x    3 root      root          1024 Sep  4 14:38 src
drwxr-xr-x    3 root      root          1024 Dec 16  1998 var
```

The `/usr/local` directory typically looks like:

```
drwxr-xr-x    3 root      root          4096 Sep 27 13:16 bin
drwxr-xr-x    2 root      root          1024 Feb  6  1996 doc
drwxr-xr-x    4 root      root          1024 Sep  3 08:07 etc
drwxr-xr-x    2 root      root          1024 Feb  6  1996 games
drwxr-xr-x    5 root      root          1024 Aug 21 19:36 include
drwxr-xr-x    2 root      root          1024 Sep  7 09:08 info
drwxr-xr-x    9 root      root          2048 Aug 21 19:44 lib
drwxr-xr-x   12 root      root          1024 Aug  2  1998 man
drwxr-xr-x    2 root      root          1024 Feb  6  1996 sbin
drwxr-xr-x   15 root      root          1024 Sep  7 09:08 share
```

and the `/usr/X11R6` directory also looks similar. What is apparent here is that all these directories contain a similar set of subdirectories. This set of subdirectories is called a *directory superstructure* or *superstructure*[1].

The superstructure will always contain a `bin` and `lib` subdirectory, but most all others are optional.

Each package will install under one of these superstructures, meaning that it will unpack many files into various subdirectories of the superstructure. A RedHat package would always install under the `/usr` or `/` superstructure, unless it is a graphical X Window System application which installs under the `/usr/X11R6` superstructure. Some very large applications may install under a `/opt/<package-name>` superstructure, and home-made packages usually install under the `/usr/local/` superstructure. **Packages almost never install files across different superstructures.**

Typically, most of the system is under `/usr`. This directory can be read only, since packages should never need to write to this directory — any writing is done under `/var` or `/tmp` (`/usr/var` and `/usr/tmp` are often just symlinked to `/var` or `/tmp` respectively). The small amount under `/` that is not part of another superstructure (usually about 40 megabytes) perform essential system administration functions. These are commands needed to bring up or repair the system in the absence of `/usr`.

The list of superstructure sub-directories and their descriptions is as follows:

`bin` *Binary executables.* Usually all `bin` directories are in the `PATH` environment variable so that the shell will search all these directories for binaries.

`sbin` *Superuser binary executables.* These are programs for system administration only. Only the super user will have these in their `PATH`.

`lib` *Libraries.* All other data needed by programs goes in here. Most packages have there own subdirectory under `lib` to store data files into. *Dynamically Linked Libraries* (*DLL*'s or `.so` files.)[2] are stored directly in `lib`.

`etc` *Etcetera.* Configuration files.

`var` *Variable data.* Data files that are continually being recreated or updated.

`doc` *Documentation.* This directory is discussed in Chapter 15.

`man` *Manual pages.* This directory is discussed in Chapter 15.

`info` *INFO pages.* This directory is discussed in Chapter 15.

`share` *Shared data.* Architecture independent files. Files that are independent of the hardware platform go in here. This allows them to be shared across different machines, even though they may have a different kind of processor altogether.

[1] To my knowledge this is a new term not previously used by UNIX administrators.

[2] Executable program code shared by more than one program in the `bin` directory to save disk space and memory.

`include` *C header files.* These are for development.

`src` *C source files.* These are sources to the kernel or locally built packages.

`tmp` *Temporary files.* A convenient place for running programs to create a file for temporarily use.

Linux on a single 1.44 megabyte floppy disk

You can get LINUX to run on a 1.44 megabyte floppy disk if you trim all unneeded files off an old Slackware distribution with a 2.0.3x kernel. You can compile a small 2.0.3x kernel to about 400 kilobytes (compressed). A file-system can be reduced to 2–3 megabytes of absolute essentials, and when compressed will fit into 1 megabyte. If the total is under 1.44 megabytes, then you have your LINUX on one floppy. The file-list might be as follows (includes all links):

```
/bin                /etc                /lib                    /sbin               /var
/bin/sh             /etc/default        /lib/ld.so              /sbin/e2fsck        /var/adm
/bin/cat            /etc/fstab          /lib/libc.so.5          /sbin/fdisk         /var/adm/utmp
/bin/chmod          /etc/group          /lib/ld-linux.so.1      /sbin/fsck          /var/adm/cron
/bin/chown          /etc/host.conf      /lib/libcurses.so.1     /sbin/ifconfig      /var/spool
/bin/cp             /etc/hosts          /lib/libc.so.5.3.12     /sbin/iflink        /var/spool/uucp
/bin/pwd            /etc/inittab        /lib/libtermcap.so.2.0.8 /sbin/ifsetup      /var/spool/uucp/SYSLOG
/bin/dd             /etc/issue          /lib/libtermcap.so.2    /sbin/init          /var/spool/uucp/ERRLOG
/bin/df             /etc/utmp           /lib/libext2fs.so.2.3   /sbin/mke2fs        /var/spool/locks
/bin/du             /etc/networks       /lib/libcom_err.so.2    /sbin/mkfs          /var/tmp
/bin/free           /etc/passwd         /lib/libcom_err.so.2.0  /sbin/mkfs.minix    /var/run
/bin/gunzip         /etc/profile        /lib/libext2fs.so.2     /sbin/mklost+found  /var/run/utmp
/bin/gzip           /etc/protocols      /lib/libm.so.5.0.5      /sbin/mkswap
/bin/hostname       /etc/rc.d           /lib/libm.so.5          /sbin/mount         /home/user
/bin/login          /etc/rc.d/rc.0      /lib/cpp                /sbin/route
/bin/ls             /etc/rc.d/rc.K                              /sbin/shutdown      /mnt
/bin/mkdir          /etc/rc.d/rc.M      /usr                    /sbin/swapoff
/bin/mv             /etc/rc.d/rc.S      /usr/adm                /sbin/swapon        /proc
/bin/ps             /etc/rc.d/rc.inet1  /usr/bin                /sbin/telinit
/bin/rm             /etc/rc.d/rc.6      /usr/bin/less           /sbin/umount        /tmp
/bin/stty           /etc/rc.d/rc.4      /usr/bin/more           /sbin/agetty
/bin/su             /etc/rc.d/rc.inet2  /usr/bin/sleep          /sbin/update        /dev/<various-devices>
/bin/sync           /etc/resolv.conf    /usr/bin/reset          /sbin/reboot
/bin/zcat           /etc/services       /usr/bin/zless          /sbin/netcfg
/bin/dircolors      /etc/termcap        /usr/bin/file           /sbin/killall5
/bin/mount          /etc/motd           /usr/bin/fdformat       /sbin/fsck.minix
/bin/umount         /etc/magic          /usr/bin/strings        /sbin/halt
/bin/bash           /etc/DIR_COLORS     /usr/bin/zgrep          /sbin/badblocks
/bin/domainname     /etc/HOSTNAME       /usr/bin/nc             /sbin/kerneld
/bin/head           /etc/mtools         /usr/bin/which          /sbin/fsck.ext2
/bin/kill           /etc/ld.so.cache    /usr/bin/grep
/bin/tar            /etc/psdevtab       /usr/sbin
/bin/cut            /etc/mtab           /usr/sbin/showmount
/bin/uname          /etc/fastboot       /usr/sbin/chroot
/bin/ping                               /usr/spool
/bin/ln                                 /usr/tmp
/bin/ash
```

Note that the `etc` directory differs slightly from a RedHat distribution. The system startup files `/etc/rc.d` are greatly simplified under Slackware.

The `/lib/modules` directory has been stripped for the creation of this floppy. `/lib/modules/2.0.36` would contain dynamically loadable kernel drivers (modules). Instead, all needed drivers are compiled into the kernel for simplicity.

At some point, creating a single floppy distribution should be attempted as an exercise. This would be most instructive to a serious system administrator. At the very least, the reader should look through all of the commands in the `bin` directories and the `sbin` directories above and browse through the man pages of any those that are unfamiliar.

The above file-system comes from the `morecram-1.3` package available at:

`http://www.obsidian.co.za/psheer/morecram-1.3.tar.gz`

an can be downloaded to give you a very useful rescue and setup disk.

Chapter 17

Unix devices

17.1 Device files

UNIX has a beautifully consistent method of allowing programs to access hardware. Under UNIX, every piece of hardware is a file. To demonstrate this, try view the file `/dev/hda`

```
less -f /dev/hda
```

`/dev/hda` is not really a file at all. When you read from it, you are actually reading directly from the first physical hard disk of your machine. `/dev/hda` is known as a device file, and all of them are stored under the `/dev` directory.

Device files allow access to hardware. If you have a sound card install and configured, you can try:

```
cat /dev/dsp > my_recording
```

Say something into your microphone and then type:

```
cat my_recording > /dev/dsp
```

Which will play out the sound through your speakers (note that this will not always work, since the recording volume may not be set correctly, nor the recording speed.)

If no programs are currently using your mouse, you can also try:

```
cat /dev/mouse
```

If you now move the mouse, the mouse protocol commands will be written directly to your screen (it will look like garbage). This is an easy way to see if your mouse is working.

At a lower level, programs that access device files do so in two basic ways:

- They read and write to the device to send and retrieve bulk data. (Much like `less` and `cat` above).
- They use the C `ioctl` (*IO Control*) function to configure the device. (In the case of the sound card, this might set mono versus stereo, recording speed etc.)

Because every kind of device that one can think of can be twisted to fit these two modes of operation (except for network cards), UNIX's scheme has endured since its inception and is considered the ubiquitous method of accessing hardware.

17.2 Block and character devices

Hardware devices can generally be categorised into random access devices like disk and tape drives, and serial devices like mouses, sound cards and terminals.

Random access devices are usually accessed in large contiguous blocks of data that are stored persistently. They are read from in discrete units (for most disks, 1024 bytes at a time). These are known as *block* devices. Doing an `ls -l /dev/hda` shows that your hard disk is a block device by the `b` on the far left of the listing:

```
brw-r-----   1 root     disk        3,   64 Apr 27   1995 /dev/hdb
```

Serial devices on the other hand are accessed one byte at a time. Data can be read or written only once. For example, after a byte has been read from your mouse, the same byte cannot be read by some other program. These are called *character* devices and are indicated by a `c` on the far left of the listing. Your `/dev/dsp` (*Digital Signal Processor* — i.e. sound card) device looks like:

```
crw-r--r--   1 root     sys        14,    3 Jul 18   1994 /dev/dsp
```

17.3 *Major* and *Minor* device numbers

Devices are divided into sets called *major device numbers*. For instance, all SCSI disks are *major number 8*. Further, each individual device has a *minor device number* like `/dev/sda` which is *minor device 0*). The major and minor device number is what identifies the device to the kernel. The file-name of the device is really arbitrary and is chosen for convenience and consistency. You can see the major and minor device number (`8,  0`) in the `ls` listing for `/dev/sda`:

```
brw-rw----   1 root     disk        8,    0 May  5   1998 /dev/sda
```

17.4 Miscellaneous devices

A list of common devices and their descriptions follows. The major numbers are shown in braces. The complete reference for Devices is the file `/usr/src/linux/Documentation/devices.txt`.

`/dev/hd??` `hd` stands for *Hard Disk*, but refers here only to *IDE* devices — i.e. common hard disks. The first letter after the `hd` dictates the physical disk drive:

`/dev/hda` **(3)** First drive, or primary master.

`/dev/hdb` **(3)** Second drive, or primary slave.

`/dev/hdc` **(22)** Third drive, or secondary master.

`/dev/hdd` **(22)** Fourth drive, or secondary slave.

When accessing any of these devices, you would be reading raw from the actual physical disk starting at the first sector of the first track, sequentially, until the last sector of the last track.

Partitions[1] are named `/dev/hda1`, `/dev/hda2` etc.indicating the first, second etc.partition on physical drive `a`.

`/dev/sd??` **(8)** `sd` stands for *SCSI Disk*, the high end drives mostly used by servers. `sda` is the first physical disk probed and so on. Probing goes by Scsi ID and has a completely different system to IDE devices. `/dev/sda1` is the first partition on the first drive etc.

`/dev/ttyS?` **(4)** These are serial devices devices numbered from `0` up. `/dev/ttyS0` is your first serial port (COM1 under DOS). If you have a multi-port card, these can go up to `32`, `64` etc.

`/dev/psaux` **(10)** PS/2 mouse.

`/dev/mouse` Is just a symlink to `/dev/ttyS0` or `/dev/psaux`. There are other mouse devices supported also.

`/dev/modem` Is just a symlink to `/dev/ttyS1` or whatever port your modem is on.

[1] With all operating systems, disk drives are divided into sections called *partitions*. A typical disk might have 2 to 10 partitions. Each partition acts as a whole disk on its own, giving the effect of having more than one disk. For instance, you might have Windows installed on one partition, and LINUX installed on another.

`/dev/cua?` **(4)** Identical to `ttyS?` but now fallen out of use.

`/dev/fd?` **(2)** *Floppy disk.* `fd0` is equivalent to your `A:` drive and `fd1` your `B:` drive. The `fd0` and `fd1` devices auto-detect the format of the floppy disk, but you can explicitly specify a higher density by using a device name like `/dev/fd0H1920` which gives you access to 1.88MB formatted 3.5 inch floppies.

See Section 18.3 on how to format these devices.

Floppy devices are named `/dev/fdlmnnnn`		
l	`0`	`A:` drive
	`1`	`B:` drive
m	`d`	"double density", "360kB" 5.25 inch
	`h`	"high density", "1.2MB" 5.25 inch
	`q`	"quad density" 5.25 inch
	`D`	"double density", "720kB" 3.5 inch
	`H`	"high density", "1.44MB" 3.5 inch
	`E`	Extra density 3.5 inch.
	`u`	Any 3.5 inch floppy. Note that `u` is now replacing `D`, `H` and `E`, thus leaving it up to the user to decide if the floppy has enough density for the format.
nnnn	`360 410 420 720 800 820 830 880 1040 1120 1200 1440 1476 1494 1600 1680 1722 1743 1760 1840 1920 2880 3200 3520 3840`	The size of the format. With `D`, `H` and `E`, 3.5 inch floppies only have devices for the sizes that are likely to work. For instance there is no `/dev/fd0D1440` because double density disks won't manage 1440kB. `/dev/fd0H1440` and `/dev/fd0H1920` are probably the ones you are most interested in.

`/dev/par?` **(6)** *Parallel port.* `/dev/par0` is your first parallel port or LPT1 under DOS.

`/dev/lp?` **(6)** *Line printer.* Identical to `/dev/par?`.

`/dev/random` *Random* number generator. Reading from this device give pseudo random numbers.

`/dev/st?` **(9)** *SCSI tape.* SCSI backup tape drive.

`/dev/zero` **(1)** Produces zero bytes, and as many of them us you need. This is useful if you need to generate a block of zeros for some reason. Use `dd` (see below) to read a specific number of zeros.

`/dev/null` **(1)** *Null device.* Reads nothing. Anything you write to the device is discarded. This is very useful for discarding output.

`/dev/pd?` *parallel port IDE disk.*

`/dev/pcd?` *parallel port ATAPI CDROM.*

`/dev/pf?` *parallel port ATAPI disk.*

`/dev/sr?` *SCSI CDROM.*

`/dev/fb?` **(29)** *Frame buffer.* This represents the kernels attempt at a graphics driver.

`/dev/cdrom` Is just a symlink to `/dev/hda`, `/dev/hdb` or `/dev/hdc`. It also my be linked to your SCSI CDROM.

`/dev/ttyI?` *ISDN Modems.*

`/dev/tty?` **(4)** *Virtual console.* This is the terminal device for the virtual console itself and is numbered `/dev/tty1` through `/dev/tty63`.

/dev/tty?? (3) and /dev/pty?? (2) Other *TTY* devices used for emulating a terminal. These are called *pseudo-TTY*'s and are identified by two lower case letters and numbers, such as ttyq3. To non-developers, these are mostly of theoretical interest.

17.5 The dd command and tricks with block devices

dd probably originally stood for *disk dump*. It is actually just like cat except it can read and write in discrete blocks. It essentially reads and writes between devices while converting the data in some way. It is generally used in one of these ways:

```
dd if=<in-file> of=<out-file> [bs=<block-size>] \
       [count=<number-of-blocks>] [seek=<output-offset>] \
       [skip=<input-offset>]

dd if=<in-file> [bs=<block-size>] [count=<number-of-blocks>] \
       [skip=<input-offset>] > <outfile>

dd of=<out-file> [bs=<block-size>] [count=<number-of-blocks>] \
       [seek=<output-offset>] < <infile>
```

dd works by specifying an input file and an output file with the if= and of= options. If the of= option is omitted, then dd writes to stdout. If the if= option is omitted, then dd reads from stdin.

To create a new RedHat boot floppy, find the boot.img file on ftp.redhat.com, and with a new floppy, do:

```
dd if=boot.img of=/dev/fd0
```

This will write the raw disk image directly to the floppy disk.

If you have ever tried to repartition a LINUX disk back into a DOS/Windows disk, you will know that DOS/Windows FDISK has bugs in it that prevent it from recreating the partition table. A quick:

```
dd if=/dev/zero of=/dev/hda bs=1024 count=10240
```

will write zeros to the first ten megabytes of your first IDE drive. This will wipe out the partition table as well as any file-system and give you a "barnd new" disk.

To zero a floppy disk is just as easy:

```
dd if=/dev/zero of=/dev/fd0 bs=1024 count=1440
```

If you have two IDE drives that are of identical size, provided that you are sure that they contain no bad sectors, you can do

```
dd if=/dev/hdc of=/dev/hdd
```

to copy the entire disk and avoid having to install an operating system from scratch. It doesn't matter what is on the original (Windows, LINUX or whatever) since each sector is identically duplicated, the new system will work perfectly.

tar can be used to backup to any device. Consider periodic backups to an ordinary IDE drive instead of a tape. Here we backup to the secondary slave:

```
tar -cvzf /dev/hdd /bin /boot /dev /etc /home /lib /sbin /usr /var
```

tar can also backup accross multiple floppy disks:

```
tar -cvMf /dev/fd0 /home/simon
```

If you don't want to see any program output, just append `> /dev/null` to the command. For example, we aren't often interested in the output of `make`[2], only the error messages:

```
make > /dev/null
```

And,

```
make >& /dev/null
```

also absorbs all error messages. `/dev/null` finds enumerable uses in shell scripting to suppress the output of a command or feed a command dummy (empty) input. `/dev/null` is a *safe* file from a security point of view, and is often used where a file is required for some feature in some configuration script, where you would like the particular feature disabled. For instance, specifying the users shell to `/dev/null` inside the password file will **certainly** prevent insecure use of a shell, and is an explicit way of saying that that account does **not** allow shell logins.

`/dev/null` can also be used to create a file containing nothing:

```
cat /dev/null > myfile
```

or alternatively, to create a file containing only zeros, try

```
dd if=/dev/zero bs=1024 count=<number-of-kilobytes> > myfile
```

17.6 Creating devices with `mknod` and `/dev/MAKEDEV`

Although all devices are listed in the `/dev` directory, you can create a device anywhere in the file system using the `mknod` command:

```
mknod [-m <mode>] <file-name> [b|c] <major-number> <minor-number>
```

The letters `b` and `c` are for creating a block or character device respectively.

To demonstrate, try

```
mknod -m 0600 ~/my-floppy b 2 0
ls -al /dev/fd0 ~/my-floppy
```

`my-floppy` can be used just like `/dev/fd0`

Note carefully the *mode* (i.e. the permissions) of `/dev/fd0`. `/dev/fd0` should be readable and writable only to `root` and to users belonging to the `floppy` group, since we obviously don't want an arbitrary user to be able to login (remotely) and write over a floppy disk.

In fact, this is the reason for having devices represented as files in the first place. UNIXfiles naturally support group access control, and therefore so also do devices.

To create devices that are missing from your `/dev` directory (some esoteric devices will not be present by default). Simply look up the device's major and minor number in `/usr/src/linux/Documentation/devices.txt` and use the `mknod` command. This is however somewhat tedious, and the script `/dev/MAKEDEV` is usually present for convenience. **You must be in the `/dev` directory before you run this script.**

Typically example usage of `MAKEDEV` is,

```
cd /dev
./MAKEDEV -v fd0
./MAKEDEV -v fd1
```

[2] `make` will be discussed later.

to create a complete set of floppy disk devices.

The man page for MAKEDEV contains more details, and explains the following:

> Note that programs giving the error "ENOENT: No such file or directory" normally means that the device file is missing, whereas "ENODEV: No such device" normally means the kernel does not have the driver configured or loaded.

Chapter 18

Partitions, file-systems, formatting and mounting

18.1 The structure of a physical disk

Cylinders, heads and sectors

Physical disks are divided into partitions[1]. Information as to how the disk is partitioned up, is stored in a *partition table*, which is a small area of the disk separate from the partitions themselves.

The physical drive itself is usually comprised of several actual disks of which both sides are used. The sides are labelled 0, 1, 2, 3 etc. and are also called *heads* because there is a magnetic head per side to do the actual reading and writing. Each side/head has tracks and each track is divided into segments called *sectors*. Each sector typically holds 512 bytes. The total amount of space on the drive in bytes is therefore:

512 * (sectors-per-track) * (tracks-per-side) * (number-of-sides).

A single track and all the tracks of the same diameter (on all the sides) are called a *cylinder*. Disks are normally talked about in terms of "cylinders and sectors" instead of "sides, tracks and sectors". Partitions are (usually) divided along cylinder boundaries. Hence disks do not have arbitrarily sized partitions; rather the size of the partition must be a multiple of the amount of data held in a single cylinder. Partitions therefore have a definite inner and outer diameter.

LBA Mode

The above system is quite straight forward except for the curious limitation that partition tables have only 10 bits to store the partition's cylinder offset. This means that no disk can have more than 1024 cylinders. This was overcome by multiplying up the number of heads in software to reduce the number of cylinders[2] hence portraying a disk of impossible proportions. The user however need never be concerned that the physical disk is completely otherwise.

Extended partitions

The partition table has room for only four partitions. To have more partitions, one of these four partitions can be divided into many smaller partitions, called *logical* partitions. The original four are then called *primary* partitions. If a primary partition is subdivided in this way, then it is know as an *extended primary* or *extended* partition. Typically, the first primary partition will be small (`/dev/hda1`, say). The second primary partition will fill the rest of the disk as an extended partition (`/dev/hda2`, say). `/dev/hda3` and `/dev/hda4`'s entries in the partition table will be left blank. The extended partition can be subdivided repeatedly to give `/dev/hda5`, `/dev/hda6` etc.

[1] See footnote on page 86
[2] Called LBA (Large Block Addressing) mode

18.2 Partitioning a new disk

A new disk will have no partition information. Typing `fdisk` will start an interactive partitioning utility.

```
fdisk /dev/hda
```

`fdisk`s your primary master.

What follows is an example of the partitioning of a new hard drive. Most distributions these days have a simpler graphical system for creating partitions, hence using `fdisk` will not be necessary at installation time. However, adding a new drive or transferring/copying a LINUX system to new hardware will require partitioning.

On UNIX, each partition has its own directory. Hence files in one directory might be stored on a different disk or a different partition to files in another directory. Typically, the `/var` directory (and all sub-directories beneath it) will be stored on a separate partition from the `/usr` directory (and all sub-directories beneath it).

Table 18.1 gives a general guideline as to how a server machine should be set up (with home computers, you can be far more liberal — most home PC's can do with a swap and `/` partition only.). When installing a new server, you distribution should allow you to customise your partitions to match this table.

Directory	Size (Megabytes)	Why?
swap	Twice the size of your RAM	This is where memory gets drawn from when you run out. It gives programs the impression that you have more RAM than you actually do, by swapping data in and out of this partition. Its slow, but it works nicely when there are a lot of programs running that are idle for most of the time, allowing their memory can be swapped to disk. Swap partitions cannot be over 128MB, but you can have many of them.
`/var`	100–1000	Here is variable data, like log files, mail spool files and your web proxy cache. If you are going to be using a web cache, either store the stuff in a separate partition/disk or make your `/var` partition huge. Also, Log files can grow to enormous sizes when there are problems and you wouldn't want a full `/var` partition to make the rest of your disk full. This is why it goes in its own partition.
`/tmp`	50	Here is temporary data. Programs access this frequently and need it to be fast. It goes in a separate partition because programs *really* need to create a temporary file sometimes, and this should not be effected by other partitions becoming full.
`/usr`	500–1500	Here is your distribution (Debian, RedHat, Mandrake etc.). It can be mounted readonly. If you have a disk whose write access can physically be disabled (like some SCSI drives), then you can put `/usr` on a separate drive. This will make for a much more secure system. Since `/usr` is stock standard, this is the partition you can most afford to loose.
`/home`	Remainder of disk	Here are your users' home directories, and any local data that this site serves (like FTP files or web pages). This is data that is most likely to fill up the partition by accident and is also the most important data to the site.
`/`	50–100	Anything not in any of the other directories is directly under your `/` directory. These are your `/bin` (5MB), `/boot` (3MB), `/dev` (100kb), `/etc` (4MB), `/lib` (20MB), `/mnt` (0), `/proc` (0) and `/sbin` (4MB) directories. They are essential for the system to startup, and contain minimal utilities for recovering the other partitions in an emergency.

Table 18.1: What directories should have their own partitions, and their partitions' sizes

If you have another operating system already installed in the first partition, you can type `p` and might see:

```
Command (m for help): p

Disk /dev/hda: 255 heads, 63 sectors, 788 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot      Start         End      Blocks   Id  System
```

```
/dev/hda1               1         312     2506108+    c  Win95 FAT32 (LBA)
```

In which case, you can just start adding partitions after it.

If you have a SCSI disk the exact same procedure applies. The only difference is that /dev/hd? changes to /dev/sd?.

```
[root@cericon /root]# fdisk /dev/hda
Device contains neither a valid DOS partition table, nor Sun or SGI disklabel
Building a new DOS disklabel. Changes will remain in memory only,
until you decide to write them. After that, of course, the previous
content won't be recoverable.


Command (m for help): p

Disk /dev/hda: 255 heads, 63 sectors, 788 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot    Start       End    Blocks   Id  System

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-788, default 1): 1
Last cylinder or +size or +sizeM or +sizeK (1-788, default 788): +80M

Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
e
Partition number (1-4): 2
First cylinder (12-788, default 12): 12
Last cylinder or +size or +sizeM or +sizeK (12-788, default 788): 788

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (12-788, default 12): 12
Last cylinder or +size or +sizeM or +sizeK (12-788, default 788): +64M

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (21-788, default 21): 21
Last cylinder or +size or +sizeM or +sizeK (21-788, default 788): +100M

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (34-788, default 34): 34
Last cylinder or +size or +sizeM or +sizeK (34-788, default 788): +200M

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (60-788, default 60): 60
Last cylinder or +size or +sizeM or +sizeK (60-788, default 788): +1500M

Command (m for help): n
Command action
```

```
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (252-788, default 252): 252
Last cylinder or +size or +sizeM or +sizeK (252-788, default 788): 788

Command (m for help): l

 0  Empty           16  Hidden FAT16    61  SpeedStor       a6  OpenBSD
  [...]
 8  AIX             4d  QNX4.x          82  Linux swap      db  CP/M / CTOS / .
 9  AIX bootable    4e  QNX4.x 2nd part 83  Linux           e1  DOS access
  [...]
12  Compaq diagnost 56  Golden Bow      a5  BSD/386         ff  BBT
14  Hidden FAT16 <3 5c  Priam Edisk

Command (m for help): t
Partition number (1-9): 5
Hex code (type L to list codes): 82
Changed system type of partition 5 to 82 (Linux swap)

Command (m for help): a
Partition number (1-10): 1

Command (m for help): p

Disk /dev/hda: 255 heads, 63 sectors, 788 cylinders
Units = cylinders of 16065 * 512 bytes

   Device Boot     Start        End     Blocks    Id  System
/dev/hda1   *          1         11      88326    83  Linux
/dev/hda2             12        788    6241252+    5  Extended
/dev/hda5             12         20      72261    82  Linux swap
/dev/hda6             21         33     104391    83  Linux
/dev/hda7             34         59     208813+   83  Linux
/dev/hda8             60        251    1542208+   83  Linux
/dev/hda9            252        788    4313421    83  Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
```

For the above partition, the kernel will give the following information at boot time:

```
Partition check:
 hda: hda1 hda2 < hda5 hda6 hda7 hda8 hda9 >
```

The < . . . > shows that partition hda2 is extended and is subdivided into five smaller partitions.

18.3 Formatting devices, partitions and floppies

Disk drives are usually read in blocks of 1024 bytes (two sectors). From the point of view of anyone accessing the device, blocks are stored consecutively — there is no need to think about cylinders or heads — so that any program can read the disk as though it were a linear tape. Try

```
less /dev/hda1
less -f /dev/hda1
```

Now a directory structure with files of arbitrary size has to be stored in this contiguous partition. This poses the problem of what to do with a file that gets deleted and leaves data "hole" in the partition, or a file that has to be split into parts because there is no single contiguous space big enough to hold it. Files also have to be

indexed in such a way that they can be found quickly (consider that there can easily be 10000 files on a system), and UNIX's symbolic/hard links and devices files also have to be stored.

To cope with this complexity, operating systems have a format for storing files called the *file-system* (fs). Like MSDOS's FAT file-system or Windows' FAT32 file-system, LINUX has a file-system called the *2nd extended file-system*, or ext2[3].

mke2fs

To create a file-system on a blank partition, the command mkfs or one of its variants is used. To create a LINUX ext2 file-system on the first partition of the primary master:

```
mkfs -t ext2 -c /dev/hda1
```

or, alternatively

```
mke2fs -c /dev/hda1
```

The -c option means to check for bad blocks by reading through the entire disk first. This is a *read-only* check and will cause unreadable blocks to be flagged as such and not be used. To do a full *read-write* check, use the badblocks command. This will write to and verify every bit in that partition. Although the -c option should always be used on a new disk, doing a full read-write test is probably pedantic. For the above partition, this would be:

```
badblocks -o blocks-list.txt -s -w /dev/hda1 88326
mke2fs -l blocks-list.txt /dev/hda1
```

Formatting floppies and removable drives

New kinds of removable devices are being released all the time. Whatever the device, the same formatting procedure is used. Most are IDE compatible, which means you can access them through /dev/hd?.

The following examples are a parallel port IDE disk drive, a parallel port ATAPI CDROM drive, a parallel port ATAPI disk drive, and your "A:" floppy drive respectively:

```
mke2fs -c /dev/pda1
mke2fs -c /dev/pcd0
mke2fs -c /dev/pf0
mke2fs -c /dev/fd0
```

Actually, using an ext2 file-system on a floppy drive wastes a lot of space. Rather use an MSDOS file-system which has less overhead, and can be read by anyone (see Section 18.3).

You often will not want to be bothered with partitioning a device that is only going to have one partition anyway. In this case you can use the whole disk as one partition. An example is a removable IDE drive as a primary slave[4]:

```
mke2fs -c /dev/hdb
```

Creating MSDOS floppies

Accessing files on MSDOS/Windows floppies is explained in Section 4.13. The command mformat A: will format a floppy, but this command merely initialises the file-system, it does not check for bad blocks or do the low level formatting necessary to reformat floppies to odd storage sizes.

[3] There are three other file-systems which may soon become standards. These are SGI's XFS, ext3fs, and reiserfs. The purpose of these is to support fast and reliable recovery in the event of a power failure. This is called *journaling* because it works by pre-writing disk writes to a separate table.

[4] *LS120* disks and *Jazz* drives as well as removable IDE brackets, are commercial examples.

There is a command called superformat from the fdutils package[5] that formats a floppy in any way that you like. It verifies that each track is working properly and compensates for variations between the mechanics of different floppy drives. To format a 3.5 inch 1440kB, 1680kB or 1920kB floppy respectively, do:

```
cd /dev
./MAKEDEV -v fd0
superformat /dev/fd0H1440
superformat /dev/fd0H1690
superformat /dev/fd0H1920
```

Note that these are "long filename" floppies (VFAT), not old 13 character filename MSDOS floppies.

Most would have only ever used a 3.5 inch floppy as a "1.44MB" floppy. In fact the disk media and magnetic head can write much more densely than this specification, allowing 24 sectors per track to be stored instead of the usual 18. This is why there is more than one device file for the same drive. Some inferior disks will however give errors when trying to format that densely — superformat will show errors when this happens.

See page 87 for how floppy devices are named, and their many respective formats.

mkswap, swapon and swapoff

The mkswap command formats a partition to be used as a swap device. For our disk:

```
mkswap -c /dev/hda5
```

-c has the same meaning as previously — to check for bad-blocks.

Once it is formatted, the kernel can be signalled to use that partition as a swap partition with

```
swapon /dev/hda5
```

and to stop usage,

```
swapoff /dev/hda5
```

Swap partitions cannot be larger than 128MB, although you can have as many of them as you like. You can swapon many different partitions simultaneously.

18.4 mounting partitions, floppies, CDROM drives and other devices

(i.e accessing the filesystem on an arbitrary disk)

The question of how to access files on an arbitrary disk (without C:, D: etc. notation, of course) is answered here.

In UNIX, there is only one root file-system that spans many disks. Different directories may actually exist on a different physical disk.

> *To bind a directory to a physical device (like a partition or a CDROM), in order that the device's file-system can be read, is called* mounting *the device.*

The mount device usually is used,

```
mount [-t <fstype>] [-o <option>] <device> <directory>
umount [-f] [device> | <directory>]
```

[5] You may have to find this package on the Internet. See Chapter ?? for how to compile and install source packages.

The -t option tells what kind of file-system it is, and can often be omitted since LINUX can auto-detect most file-systems. <fstype> can be one of adfs, affs, autofs, coda, coherent, devpts, efs, ext2, hfs, hpfs, iso9660, minix, msdos, ncpfs, nfs, ntfs, proc, qnx4, romfs, smbfs, sysv, ufs, umsdos, vfat, xenix or xiafs. The most common ones are discussed below. The -o option is not usually used. See the mount(8) page for all possible options.

Put your distribution CDROM disk into your CDROM drive and mount it with,

```
ls /mnt/cdrom
mount -t iso9660 -o ro /dev/hdb /mnt/cdrom
```

(Your CDROM might be /dev/hdc or /dev/hdd however — in this case you should make a soft link /dev/cdrom pointing to the correct device.) Now cd to your /mnt/cdrom directory. You will notice that it is no longer empty, but "contains" the CDROM's files. What is happening is that the kernel is redirecting all lookups from the directory /dev/cdrom to read from the CDROM disk. You can browse around these files as thought they were already copied onto your hard drive. This is what makes UNIX cool.

When you are finished with the CDROM *unmount* it with,

```
umount /dev/hdb
eject /dev/hdb
```

Instead of using mtools, you could mount the floppy disk with:

```
mkdir /mnt/floppy
mount -t vfat /dev/fd0 /mnt/floppy
```

or, for older MSDOS floppies:

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

Before you eject the floppy, it is essential to

```
umount /dev/fd0
```

in order that cached data is committed to the disk. Failing to umount a floppy before ejecting will probably cause its file-system to be corrupted.

mounting Windows and NT partitions

Mounting a Windows partition can also be done with the vfat file-system, and NT partitions (read only) with the ntfs file-system. VAT32 is auto-detected and supported. For example,

```
mkdir /windows
mount -t vfat /dev/hda1 /windows
mkdir /nt
mount -t ntfs /dev/hda2 /nt
```

18.5 Checking and repairing file-systems with fsck

fsck stands for *file system check*. fsck scans the file-system, reporting and fixing errors. Errors would normally occur because the kernel halted before umounting the file-system. In this case, it may have been in the middle of a write operation which left the file-system in an *incoherent* state. This usually would happen because of a power failure.

It is used as follows:

```
fsck [-V] [-a] [-t <fstype>] <device>
```

-V means to produce verbose output. -a means to check the file-system non-interactively — meaning to not ask the user before trying to make any repairs. This is what you would normally do with LINUX if you don't know a whole lot about the ext2 file-system:

```
fsck -a -t ext2 /dev/hda1
```

although the -t option can be omitted as LINUX auto-detects the file-system. Note that you cannot run fsck on a mounted file-system.

fsck actually just runs a program specific to that file-system. In the case of ext2, the command e2fsck (also known as fsck.ext2) is run. Do a e2fsck(8) to get exhaustive details.

18.6 Automatic mounting at boot time with /etc/fstab

Above, manual mounts are explained for new and removable disks. It is of course necessary for file-systems to be automatically mounted at boot time. What gets mounted and how is specified in the configuration file /etc/fstab.

It will usually look something like this for the disk we partitioned above:

```
/dev/hda1               /               ext2    defaults        1 1
/dev/hda6               /tmp            ext2    defaults        1 2
/dev/hda7               /var            ext2    defaults        1 2
/dev/hda8               /usr            ext2    defaults        1 2
/dev/hda9               /home           ext2    defaults        1 2
/dev/hda5               swap            swap    defaults        0 0
/dev/fd0                /mnt/floppy     auto    noauto          0 0
/dev/cdrom              /mnt/cdrom      iso9660 noauto,ro       0 0
none                    /proc           proc    defaults        0 0
none                    /dev/pts        devpts  mode=0622       0 0
```

For the moment we are interested in the first six lines only. The first three fields (columns) dictates the partition, the directory where it is to be mounted, and the file-system type respectively. The fourth field gives options (the -o option to mount).

The fifth field tells whether the file-system contains real files. It is used by the dump to decide if it can should be backed up. This is not commonly used.

The last field tells the order in which an fsck should be done on the partitions. The / partition should come first with a 1, and all other partitions should come directly after. By placing 2's everywhere else, it ensures that partitions on different disks can be checked in parallel, which speeds things up slightly at boot time.

The floppy and cdrom entries allow you to use an abbreviated form of the mount command. mount will just look up the corresponding directory and file-system type from /etc/fstab. Try

```
mount /dev/cdrom
```

proc is a kernel info database that looks like a file-system. For example /proc/cpuinfo is not any kind of file that actually exists on a disk somewhere. Try cat /proc/cpuinfo.

Many programs use /proc to get information on the status and configuration of your machine. More on this will be discussed in Section ??.

The devpts file-system is another pseudo-file-system that generates terminal master/slave pairs for programs. This is mostly of concern to developers.

18.7 RAM and loopback devices

A *RAM* device is a block device that can be used as a disk, but really points to a physical area of RAM.

A *loopback* device is a block device that can be used as a disk, but really points to an ordinary file somewhere.

If your imagination isn't already running wild, consider creating a floppy disk with file-system, files and all, without actually having a floppy disk; and then writing the results to a file that can be dd'd to a floppy at any time. You can do this with *loopback* and *RAM* devices.

You can have a whole other LINUX system inside a 500MB file on a Windows partition and boot into it — thus obviating having to repartition a Windows machine just to run LINUX.

Formatting a floppy inside a file

The operations are quite trivial. To create an MSDOS floppy inside a 1440kB *file*, do:

```
dd if=/dev/zero of=~/file-floppy count=1440 bs=1024
losetup /dev/loop0 ~/file-floppy
mke2fs /dev/loop0
mkdir ~/mnt
mount /dev/loop0 ~/mnt
ls -al ~/mnt
```

When you are finished copying the files that you want into /mnt, merely

```
umount ~/mnt
losetup -d /dev/loop0
```

To dump the file-system to a floppy,

```
dd if=~/file-floppy of=/dev/fd0 count=1440 bs=1024
```

A similar procedure for RAM devices

```
dd if=/dev/zero of=/dev/ram0 count=1440 bs=1024
mke2fs /dev/ram0
mkdir ~/mnt
mount /dev/ram0 ~/mnt
ls -al ~/mnt
```

When you are finished copying the files that you want into /mnt, merely

```
umount ~/mnt
```

To dump the file-system to a floppy or file respectively:

```
dd if=/dev/ram0 of=/dev/fd0 count=1440 bs=1024
dd if=/dev/ram0 of=~/file-floppy count=1440 bs=1024
```

Chapter 19

Trivial introduction to C

The **C** programming language was invented for the purposes of writing an operating system that could be recompiled (ported) to different hardware platforms (different CPU's). It is hence also the first choice for writing any kind of application that has to communicate efficiently with the operating system.

Many people who don't program in **C** very well think of **C** as an arbitrary language out of many. This point should be made at once: **C** is the fundamental basis of all computing in the world today. UNIX, Microsoft Windows, office suites, web browsers and device drivers are all written in **C**. 99% of your time spent at a computer is probably spent inside a **C** application[1].

There is also no replacement for **C**. Since it fulfils its purpose without much flaw, there will never be a need to replace it. *Other languages may fulfil other purposes, but **C** fulfils its purpose most adequately.* For instance, all future operating systems will probably be written in **C** for a long time to come.

It is for these reasons that your knowledge of UNIX will never be complete until you can program in **C**.

19.1 C fundamentals

19.1.1 The simplest C program

A simple **C** program is:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    printf ("Hello World!\n");
    return 3;
}
```

Save this program in a file `hello.c`. You can compile this program with the command

```
gcc -Wall -o hello hello.c
```

the `-o hello` option tells `gcc`[2] to produce the binary file `hello` instead of the default binary file name `a.out`[3]. The `-Wall` option means to report `all` `W`arnings during the compilation. This is not strictly necessary but most helpful for correcting possible errors in your programs.

Then run the program with

```
./hello
```

[1] C++ is also quite popular. It is, however, not as fundamental to computing, although it is more suitable in many situations.
[2] GNU C Compiler. `cc` on other UNIX systems.
[3] Called `a.out` out of historical reasons.

Previously you should have familiarised yourself with bash functions (See Section 6.6). In **C** *all* code is inside a function. The first function to be called (by the operating system) in the main function.

Type echo $? to see the return code of the program. You will see it is 3, indicating the return value of the main function.

Other things to note are the " on either side of the string to be printed. Quotes are required around string literals. Inside a string literal \n *escape sequence* indicates a newline character. ascii(7) shows some other escape sequences. You can also see a proliferance of ; everywhere in a **C** program. Every statement in **C** is separated by a ; unlike with shell scripts where a ; is optional.

Now try:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    printf ("number %d, number %d\n", 1 + 2, 10);
    exit (3);
}
```

printf may be thought of as the command to send output to the terminal. It is also what is known as a *standard C library function*. In other words, it is specified that a **C** implementation should always have the printf function and that it should behave in a certain way.

The %d indicates that a *decimal* should go in at that point in the text. The number to be substituted will be the first *argument* to the printf function after the string literal — i.e. the 1 + 2. The next %d is substituted with the second argument — i.e. the 10. The %d is known as a *format specifier*. It essentially *converts* an integer number into a decimal representation.

19.1.2 Variables and types

With bash you could use a variable anywhere anytime, and the variable would just be blank if it was never used before. In **C** you have to tell the compiler what variables you are going to need before each block of code.

This is done with a variable declaration:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    int x;
    int y;
    x = 10;
    y = 2;
    printf ("number %d, number %d\n", 1 + y, x);
    exit (3);
}
```

The int x is a variable declaration. It tells the program to reserve space for one int*eger* variable and that it will later be referred to as x. int is the *type* of the variable. x = 10 assigned the variable with the value 10. There are types for each of the kind of numbers you would like to work with, and format specifiers to convert them for printing:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    char a;
    short b;
    int c;
    long d;
    float e;
    double f;
    long double g;
```

```
    a = 'A';
    b = 10;
    c = 10000000;
    d = 10000000;
    e = 3.14159;
    f = 10e300;
    g = 10e300;
    printf ("%c, %hd, %d, %ld, %f, %f, %Lf\n", a, b, c, d, e, f, g);
    exit (3);
}
```

You will notice that `%f` is used for both `float`*s and* `double`*s*. This is because `float`*s* are always converted to `double`*s* during an operating like this. Also try replacing `%f` with `%e` to print in exponential notation — i.e. less significant digits.

19.1.3 Functions

Functions are implemented as follows:

```
#include <stdlib.h>
#include <stdio.h>

void mutiply_and_print (int x, int y)
{
    printf ("%d * %d = %d\n", x, y, x * y);
}

int main (int argc, char *argv[])
{
    mutiply_and_print (30, 5);
    mutiply_and_print (12, 3);
    exit (3);
}
```

Here we have a non-main function *called* by the `main` function. The function is first *declared* with

```
void mutiply_and_print (int x, int y)
```

This declaration states the return value of the function (`void` for no return value); the function name (`mutiply_and_print`) and then the *arguments* that are going to be passed to the function. The numbers passed to the function are given their own names, `x` and `y`, and are converted to the type of `x` and `y` before being passed to the function — in this case, `int` and `int`. The actual **C** code of which the function is comprised goes between curly braces { and }.

In other word, the above function is the same as:

```
void mutiply_and_print ()
{
    int x;
    int y;
    x = <first-number-passed>
    y = <second-number-passed>
    printf ("%d * %d = %d\n", x, y, x * y);
}
```

(Note that this is not permissable **C** code.)

19.1.4 `for`, `while`, `if` and `switch` statements

As with shell scripting, we have the `for`, `while` and `if` statements:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
```

```
    int x;

    x = 10;

    if (x == 10) {
        printf ("x is exactly 10\n");
        x++;
    } else if (x == 20) {
        printf ("x is not equal to 20\n");
    } else {
        printf ("No, x is not equal to 10 or 20\n");
    }

    if (x > 10) {
        printf ("Yes, x is more than 10\n");
    }

    while (x > 0) {
        printf ("x is %d\n", x);
        x = x - 1;
    }

    for (x = 0; x < 10; x++) {
        printf ("x is %d\n", x);
    }

    switch (x) {
        case 9:
            printf ("x is nine\n");
            break;
        case 10:
            printf ("x is ten\n");
            break;
        case 11:
            printf ("x is eleven\n");
            break;
        default:
            printf ("x is huh?\n");
            break;
    }

    return 0;
}
```

It is easy to see the format that these take, although they are vastly different from shell scripts. **C** code works in *statement blocks* between curly braces, in the same way that shell scripts have `do`'s and `done`'s.

Note that with most programming languages when we want to add `1` to a variable we have to write, say `x = x + 1`. In **C** the abbreviation `x++` is used, meaning to *increment* a variable by `1`.

The `for` loop takes three statements between `( ... )`. These are, a statement to start things off, a comparison, and a statement to be executed everytime after the statement block. The statement block after the `for` is executed until the comparison is untrue.

The `switch` statement is like `case` in shell scripts. `switch` considers the argument inside its `( ... )` and decides which `case` line to jump to. In this case it will obviously be `printf ("x is ten\n");` because `x` was 10 when the previous `for` loop exited. The `break` tokens means we are done with the `switch` statement and that execution should continue from Line 46.

Note that in **C** the comparison `==` is used instead of `=`. `=` means to assign a value to a variable, while `==` is an *equality operator*.

19.1.5 Strings, arrays and memory allocation

You can define a list of numbers with:

```
int y[10];
```

This is called an *array*:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    int x;
    int y[10];
    for (x = 0; x < 10; x++) {
        y[x] = x * 2;
    }
    for (x = 0; x < 10; x++) {
        printf ("item %d is %d\n", x, y[x]);
    }
    return 0;
}
```

If an array is of type `char`*acter* then it is called a *string*:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    int x;
    char y[11];
    for (x = 0; x < 10; x++) {
        y[x] = 65 + x * 2;
    }
    for (x = 0; x < 10; x++) {
        printf ("item %d is %d\n", x, y[x]);
    }
    y[10] = 0;
    printf ("string is %s\n", y);
    return 0;
}
```

Note that a string has to be *null-terminated*. This means that the last character must be a zero. The code `y[10] = 0` sets the eleventh item in the array to zero. This also means that strings need to be one `char` longer than you would think.

(Note that the first item in the array is `y[0]`, not `y[1]`, like some other programming languages.)

In the above example, the line `char y[11]` reserved 11 bytes for the string. Now what if you want a string of 100000 bytes? **C** allows you to *allocate memory* for your 100k which means requesting memory from the kernel. Any non-trivial program will allocate memory for itself and there is no other way of getting getting large blocks of memory for your program to use. Try:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    int x;
    char *y;
    y = malloc (11);
    printf ("%ld\n", y);
    for (x = 0; x < 10; x++) {
        y[x] = 65 + x * 2;
    }
    y[10] = 0;
    printf ("string is %s\n", y);
    free (y);
    return 0;
}
```

The declaration `char *y` would be new to you. It means to declare a variable (a number) called `y` that *points* to a memory location. The `*` (*asterix*) in this context means *pointer*. Now if you have a machine with perhaps 256 megabytes of RAM + swap, then `y` will have a range of about this much. The numerical value of `y` is also printed with `printf ("%ld\n", y);`, but is of no interest to the programmer.

When finished using memory it should be given back to the operating system. This is done with `free`. Programs that don't `free` all the memory they allocate are said to *leak* memory.

Allocating memory often requires you to perform a calculation to determine the amount of memory required. In the above case we are allocating the space of 11 `char`'s. Since each `char` is really a single byte, this presents no problem. But what if we were allocating 11 `int`'s? An `int` on a PC is 32 bits — four bytes. To determine the size of a type, we use the `sizeof` keyword:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    int a;
    int b;
    int c;
    int d;
    int e;
    int f;
    int g;
    a = sizeof (char);
    b = sizeof (short);
    c = sizeof (int);
    d = sizeof (long);
    e = sizeof (float);
    f = sizeof (double);
    g = sizeof (long double);
    printf ("%d, %d, %d, %d, %d, %d, %d\n", a, b, c, d, e, f, g);
    return 0;
}
```

Here you can see the number of bytes required by all of these types. Now we can easily allocate arrays of things other than `char`.

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    int x;
    int *y;
    y = malloc (10 * sizeof (int));
    printf ("%ld\n", y);
    for (x = 0; x < 10; x++) {
        y[x] = 65 + x * 2;
    }
    for (x = 0; x < 10; x++) {
        printf ("%d\n", y[x]);
    }
    free (y);
    return 0;
}
```

On many machines an `int` is four bytes (32 bits), but you should never assume this. **Always use the `sizeof` keyword to allocate memory.**

19.1.6 String operations

C programs probably do more string manipulation than anything else. Here is a program that divides a sentence up into words:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[])
{
    int length_of_word;
    int i;
```

```
    int length_of_sentace;
    char p[256];
    char *q;

    strcpy (p, "hello there, my name is fred.");

    length_of_sentace = strlen (p);

    length_of_word = 0;

    for (i = 0; i <= length_of_sentace; i++) {
        if (p[i] == ' ' || i == length_of_sentace) {
            q = malloc (length_of_word + 1);
            strncpy (q, p + i - length_of_word, length_of_word);
            q[length_of_word] = 0;
            printf ("word: %s\n", q);
            free (q);
            length_of_word = 0;
        } else {
            length_of_word = length_of_word + 1;
        }
    }
    return 0;
}
```

Here we introduce three more *standard C library functions*. `strcpy` stands for `string copy`. It copies memory from one place to another. Line 13 of this program copies text *into* the `char`acter array `p`, which is called the *target* of the copy.

`strlen` stands for `string length`. It determines the length of a string, which is just a count of the number of `char`acters up to the null character.

We need to loop over the length of the sentence. The variable `i` indicates the current position in the sentence.

Line 20 says that if we find a character 32 (denoted by `' '`) we know we have reached a word boundary. We also know that the end of the sentence is a word boundary even though there may not be a space there. The token `||` means **OR**. At this point we can allocate memory for the current word, and copy the word into that memory. The `strncpy` function is useful for this. It copies a string, but only up to a limit of `length_of_word` characters (the last argument). Like `strcpy`, the first argument is the target, and the second argument is the place to copy from.

To calculate the position of the start of the last word, we use `p + i - length_of_word`. This means that we are adding `i` to the memory location `p` and then going back `length_of_word` counts thus pointing `strncpy` to the exact position.

Finally, we null terminate the string on Line 23. We can then print `q`, `free` the used memory, and begin with the next word.

To get a complete list of string operations, see `string`(3).

19.1.7 File operations

Under most programming languages, file operations involve three steps: *opening* a file, *reading* or *writing* to the file, and then *closing* the file. The command `fopen` is commonly uses to tell the operating system that you are ready to begin working with a file:

The following program opens a file and spits it out on the terminal:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[])
{
    int c;
    FILE *f;

    f = fopen ("test.c", "r");
    for (;;) {
```

```
        c = fgetc (f);
        if (c == -1)
            break;
        printf ("%c", c);
    }
    fclose (f);
    return 0;
}
```

A new type is presented here: `FILE *`. It is a file operations variable that has to be *initialised* with `fopen` before we can use it. The `fopen` function takes two arguments: the first is the name of the file and the second is string explaining *how* we want to open the file — in this case `"r"` means *reading* from the start of the file. Other options are `"w"` for *writing* and several more described in `fopen`(3).

The command `fgetc` *gets* a character from the file. It retrieves consecutive bytes from the file until it reaches the end of the file, where it returns a `-1`. The `break` statement indicates to immediately terminate the `for` loop, whereupon execution will continue from Line 17. `break` statements can appear inside `while` loops as well.

You will notice that the `for` loop is empty. This is allowable **C** code and means to loop forever.

Some other file functions are `fread`, `fwrite`, `fputc`, `fprintf` and `fseek`. See `fwrite`(3), `fputc`(3), `fprintf`(3) and `fseek`(3).

19.1.8 Reading command-line arguments inside C programs

Up until now, you are probably wondering what the `(int argc, char *argv[])` are for. These are the command-line arguments passed to the program by the shell. `argc` is the number of command-line arguments and `argv` is an array of strings of each argument. Printing them out is easy:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main (int argc, char *argv[])
{
    int i;
    for (i = 0; i < argc; i++) {
        printf ("argument %d is %s\n", i, argv[i]);
    }
    return 0;
}
```

19.1.9 A more complicated example

Here we put this altogether in a program that reads in lots of files and dumps them as words. Some new things in the following program are: `!=` is the inverse of `==`. It tests if *not-equal-to*; `realloc` *reallocates* memory — it resizes an old block of memory so that any bytes of the old block are preserved; `\n`, `\t` mean the newline character, 10, or the tab character, 9, respectively.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void word_dump (char *filename)
{
    int length_of_word;
    int amount_allocated;
    char *q;
    FILE *f;
    int c;

    c = 0;

    f = fopen (filename, "r");

    length_of_word = 0;

    amount_allocated = 256;
    q = malloc (amount_allocated);

    while (c != -1) {
        if (length_of_word >= amount_allocated) {
            amount_allocated = amount_allocated * 2;
            q = realloc (q, amount_allocated);
        }

        c = fgetc (f);
```

```
        q[length_of_word] = c;

        if (c == -1 || c == ' ' || c == '\n' || c == '\t') {
            if (length_of_word > 0) {
                q[length_of_word] = 0;
                printf ("%s\n", q);
            }
            amount_allocated = 256;
            q = realloc (q, amount_allocated);
            length_of_word = 0;
        } else {
            length_of_word = length_of_word + 1;
        }
    }

    fclose (f);
}

int main (int argc, char *argv[])
{
    int i;

    if (argc < 2) {
        printf ("Usage:\n\twordsplit <filename> ...\n");
        exit (1);
    }

    for (i = 1; i < argc; i++) {
        word_dump (argv[i]);
    }

    return 0;
}
```

This program is more complicated than you might immediately expect. Reading in a file where we **know** that a word will never exceed 30 characters is simple. But what if we have a file that contains some words that are 100000 characters long? GNU programs are expected to behave correctly under these circumstances.

To cope with normal as well as extreme circumstances, we assume to start off with that a word will never be more than 256 characters. If it appears that the word is growing passed 256 characters, we `reallocate` the memory space to double its size (Line 24 amd 25). When we start with a new word, we can free up memory again, so we `realloc` back to 256 again (Line 36 and 37). In this way are only use the minimum amount of memory at each point in time.

We have hence created a program that can work efficiently with a 100 Gigabyte file just as easily as with a 100 byte file. *This is part of the art of* ***C*** *programming.*

Experiences **C** programmers may actually scoff at the above listing because it really isn't as "minimalistic" as you may be able to write it with more experience. In fact it is really a truly excellent listing for the simple reason that, firstly, it is easy to understand, and secondly, it is an efficient algorithm (albeit not optimal). *Readability in* ***C*** *is your first priority — it is imperative that what you do is* ***obvious*** *to anyone reading the code.*

19.1.10 `#include` and prototypes

At the start of each program will be one or more `#include` statements. These tell the compiler to read in another **C** program. Now "raw" **C** does not have a whole lot in the way of protecting against errors: for example the `strcpy` function could just as well be used with one, three or four arguments, and the **C** program would still compile. It would however reek havoc with the internal memory and cause the program to crash. These other `.h` **C** programs are called *header* files that contain templates for how functions are meant to be called. Every function you might like to use is contained in one or other template file. The templates are called *function prototypes*.

A function prototype is written the same as the function itself, but without the code. A function prototype for `word_dump` would simply be:

```
void word_dump (char *filename);
```

The trailing `;` is essential and distinguishes a function from a function prototype.

After a function prototype, any attempt to use the function in a way other than intended — say, passing to few arguments or arguments of the wrong type — will be met with fierce opposition from `gcc`.

You will notice that the `#include <string.h>` appeared when we started using `str`ing operations. Recompiling these programs without the `#include <string.h>` line give the warning message:

```
test.c:21: warning: implicit declaration of function `strncpy'
```

Which is quite to the point.

The function prototypes give a clear definition of how every function is to be used. man pages will always first state the function prototype so that you are clear on what arguments are to be passed, and what types they should have.

19.1.11 C comments

A **C** comment is denoted with /* <comment lines> */. Anything between the /* and */ is ignored and can span multiple lines. Every function should be commented, and all non-obvious code should be commented. It is a good rule that a program that *needs* lots of comments to explain it is *badly written*. Also, never comment the obvious and explain *why* you do things rather that *what* you are doing. It is advisable **not** to make pretty graphics between each function, so rather:

```
/* returns -1 on error, takes a positive integer */
int sqr (int x)
{
    <...>
```

than

```
/*****************************----SQR----*********************************
 *                 x = argument to make the square of                    *
 *    return value  =                                                    *
 *                          -1 (on error)                                *
 *                          square of x (on success)                     *
 ************************************************************************/
int sqr (int x)
{
    <...>
```

which is liable to give people nausea. Under C++, the additional comment // is allowed, which ignores everything between the // and the end of the line. It is accepted under gcc, but should not be used unless you really are programming in C++. In addition, programmers often "comment out" lines by placing an #if 0 ... #endif around them, which really does exactly the same thing as a comment (see Section 19.1.12), but allows you to comment out comments as well eg:

```
    int x;
    x = 10;
#if 0
    printf ("debug: x is %d\n", x);     /* print debug information */
#endif
    y = x + 10;
    <...>
```

comments out Line 4.

19.1.12 #define and #if — C macros

Anything starting with a # is not actually **C**, but a **C** *preprocessor directive*. A **C** program is first run through a *preprocessor* which removes all spurious junk, like comments and #include statements. **C** programs can be made much more readable by defining *macros* instead of literal values. For instance:

```
#define START_BUFFER_SIZE 256
```

in our example program #defines the text START_BUFFER_SIZE to be the text 256. Thereafter wherever in the **C** program we have a START_BUFFER_SIZE, the text 256 will be seen by the compiler, and we can use START_BUFFER_SIZE instead. This is a much *cleaner* way of programming, because, if say we would like to change the 256 to some other value, we only need to change it in one place. START_BUFFER_SIZE is also more meaningful than a number, making the program more readable.

Whenever you have a *literal constant* like 256, you should replace it with a macro defined near the top of your program.

You can also check for the existence of macros with the `#ifdef` and `#ifndef` directive. `#` directives are really a programming language all on their own:

```
/* Set START_BUFFER_SIZE to fine tune performance before compiling: */
#define START_BUFFER_SIZE 256
/* #define START_BUFFER_SIZE 128 */
/* #define START_BUFFER_SIZE 1024 */
/* #define START_BUFFER_SIZE 16384 */

#ifndef START_BUFFER_SIZE
#error This code did not define START_BUFFER_SIZE. Please edit
#endif

#if START_BUFFER_SIZE <= 0
#error Wooow! START_BUFFER_SIZE must be greater than zero
#endif

#if START_BUFFER_SIZE < 16
#warning START_BUFFER_SIZE to small, program may be inefficient
#elif START_BUFFER_SIZE > 65536
#warning START_BUFFER_SIZE to large, program may be inefficient
#else
/* START_BUFFER_SIZE is ok, do not report */
#endif

void word_dump (char *filename)
{
    <...>
    amount_allocated = START_BUFFER_SIZE;
    q = malloc (amount_allocated);
    <...>
```

19.2 C Libraries

We made reference to the Standard **C** Library. The **C** language on its own does almost nothing; everything useful is an external function. External functions are grouped into libraries. The Standard **C** Library is the file `/lib/libc.so.6`. To list all the **C** library functions, do:

```
nm /lib/libc.so.6
nm /lib/libc.so.6 | grep ' T ' | cut -f3 -d' ' | grep -v '^_' | sort -u | less
```

many of these have `man` pages, however some will have no documentation and require you to read the comments inside the header files. It is better not to use functions unless you are sure that they are *standard* functions in the sense that they are common to other systems.

To create your own library is simple. Lets say we have two files that contain functions that we would like to create a library out of, `simple_math_sqrt.c`,

```
#include <stdlib.h>
#include <stdio.h>

static int abs_error (int a, int b)
{
    if (a > b)
        return a - b;
    return b - a;
}

int simple_math_isqrt (int x)
{
    int result;
    if (x < 0) {
        fprintf (stderr, "simple_math_sqrt: taking the sqrt of a negative number\n");
        abort ();
    }
    result = 2;
    while (abs_error (result * result, x) > 1) {
        result = (x / result + result) / 2;
```

```
    }
    return result;
}
```

and, simple_math_pow.c

```
#include <stdlib.h>
#include <stdio.h>

int simple_math_ipow (int x, int y)
{
    int result;
    if (x == 1 || y == 0)
        return 1;
    if (x == 0 && y < 0) {
        fprintf (stderr, "simple_math_pow: raising zero to a negative power\n");
        abort ();
    }
    if (y < 0)
        return 0;
    result = 1;
    while (y > 0) {
        result = result * x;
        y = y - 1;
    }
    return result;
}
```

We would like to call the library simple_math. It is good practice to name all the functions in the library simple_math_*??????*. The function abs_error is not going to be used outside of the file simple_math_sqrt.c and hence has the keyword static in front of it, meaning that it is a *local* function.

We can compile the code with:

```
gcc -Wall -c simple_math_sqrt.c
gcc -Wall -c simple_math_pow.c
```

The -c option means the *compile only*. The code is not turned into an executable. The generated files are simple_math_sqrt.o and simple_math_pow.o. These are called o*bject* files.

We now need to *archive* these files into a library. We do this with the ar command (a predecessor to tar):

```
ar libsimple_math.a simple_math_sqrt.o simple_math_pow.o
ranlib libsimple_math.a
```

The ranlib command indexes the archive.

The library can now be used. Create a file test.c:

```
#include <stdlib.h>
#include <stdio.h>

int main (int argc, char *argv[])
{
    printf ("%d\n", simple_math_ipow (4, 3));
    printf ("%d\n", simple_math_isqrt (50));
    return 0;
}
```

and run:

```
gcc -Wall -c test.c
gcc -o test test.o -L. -lsimple_math
```

The first command compiles the file test.c into test.o, while the second function is called *linking* the program, which assimilates test.o and the libraries into a single executable. The option L. means to look in the current directory for any libraries (usually only /lib and /usr/lib are searched). The option -lsimple_math means to assimilate the library libsimple_math.a (lib and .a are added automatically).

We can also create a header file simple_math.h for using the library.

```
/* calculates the integer square root, aborts on error */
int simple_math_isqrt (int x);

/* calculates the integer power, aborts on error */
int simple_math_ipow (int x, int y);
```

Add the line `#include "simple_math.h"` to the top of `test.c`:

```
#include <stdlib.h>
#include <stdio.h>
#include "simple_math.h"
```

This will get rid of the `implicit declaration of function` warning messages. Usually `#include <simple_math.h>` would be used, but here this is a header file in the current directory — our *own* header file — and this is where we use `"simple_math.h"` instead of `<simple_math.h>`.

19.3 C projects — Makefiles

Now what if you make a small change to one of the files (as you are likely to do very often when developing)? You could script the process of compiling and linking, but the script would build everything, and not just the changed file. What we really need is a utility that only recompiles object files whose sources have changed: `make` is such a utility.

`make` is a program that looks inside a `Makefile` in the current directory then does a lot of compiling and linking. `Makefile`s contain lists of rules and *dependencies* describing how to build a program.

Inside a `Makefile` you need to state a list of *what-depends-on-what* dependencies that `make` can work through, as well as the shell commands needed to achieve each goal.

Our first (last?) *dependency* in the process of completing the compilation is that `test` *depends-on* both the library, `libsimple_math.a`, and the object file, `test.o`. In `make` terms we create a `Makefile` line that looks like:

```
test:   libsimple_math.a test.o
```

meaning simply that the files `libsimple_math.a test.o` must exist and be updated before `test`. `test:` is called a `make` *target*. Beneath this line, we also need to state how to build `test`:

```
	gcc -Wall -o $@ test.o -L. -lsimple_math
```

The `$@` means the name of the target itself which is just substituted with `test`. **Note that the space before the `gcc` is a tab character and not 8 space characters.**

The next dependency is that `libsimple_math.a` depends on `simple_math_sqrt.o simple_math_pow.o`. Once again we have a dependency, along with a shell script to build the target. The full `Makefile` *rule* is:

```
libsimple_math.a: simple_math_sqrt.o simple_math_pow.o
	rm -f $@
	ar rc $@ simple_math_sqrt.o simple_math_pow.o
	ranlib $@
```

Note again that the left margin consists of a single tab character and not spaces.

The final dependency is that the files `simple_math_sqrt.o` and `simple_math_pow.o` depend on the files `simple_math_sqrt.c` and `simple_math_pow.c`. The requires two `make` target rules, but `make` has a short way of stating such a rule for where there are many **C** source files,

```
.c.o:
	gcc -Wall -c -o $*.o $<
```

which means that any `.o` files needed can be built from a `.c` file of a similar name using the command `gcc -Wall -c -o $*.o $<`. `$*.o` means the name of the object file and `$<` means the name of the file that `$*.o` depends on, one at a time.

Putting it all together

Makefiles can in fact have their rules put in any order, so its best to state the most obvious rules first for readability.

There is also a rule you should always state at the outset:

```
all:    libsimple_math.a test
```

The all: target is the rule that make tries to satisfy when make is run with no command-line arguments. This just means that libsimple_math.a and test are the last two files to be built, i.e. the top-level dependencies.

Makefiles also have their own form of environment variables, like shell scripts. You can see that we have used the text simple_math in three of our rules. It makes sense to define a macro for this so that if we can easily change to a different library name.

Our final Makefile is:

```
# Comments start with a # (hash) character like shell scripts.
# Makefile to build libsimple_math.a and test program.
# Paul Sheer <psheer@obsidian.co.za> Sun Mar 19 15:56:08 2000

OBJS    = simple_math_sqrt.o simple_math_pow.o
LIBNAME = simple_math
CFLAGS  = -Wall

all:    lib$(LIBNAME).a test

test:   lib$(LIBNAME).a test.o
        gcc $(CFLAGS) -o $@ test.o -L. -l${LIBNAME}

lib$(LIBNAME).a: $(OBJS)
        rm -f $@
        ar rc $@ $(OBJS)
        ranlib $@

.c.o:
        gcc $(CFLAGS) -c -o $*.o $<

clean:
        rm -f *.o *.a test
```

We can now easily type

```
make
```

in the current directory to cause everything to be built.

You can see we have added an additional disconnected target clean:. Targets can be run explictly on the command-line like:

```
make clean
```

which removes all built files.

Makefiles have far more uses than just building **C** programs. Anything that needs to be built from sources can employ a Makefile to make things easier.

19.4 Building dynamically loadable libraries — .so files

Chapter 20

Introduction to IP

20.1 Internet Communication

IP stands for *Internet Protocol*. It is the method by which data gets transmitted over the Internet. At a hardware level, network cards are capable of transmitting *packets* (also called *datagrams*) of data between one another. A packet contains a small block of say, 1 kilobyte of data. (In contrast to serial lines which transmit continuously.) All Internet communication occurs via transmission of packets, which travel intact between machines on either side of the world.

Each packet contains a header preceding the data of 24 bytes or more. Hence slightly more than the said 1 kilobyte of data would be found on the wire. When a packet is transmitted, the header would obviously contain the destination machine. Each machine is hence given a unique *IP address* — a 32 bit number. There are no machines on the Internet that do not have an IP address.

The header actually looks as follows:

Bytes	Description
0	bits 0-3: Version, bits 4-7: Internet Header Length (IHL)
1	Type of service (TOS)
2 to 3	Length
4 to 5	Identification
6 to 7	bits 0-3: Flags, bits 4-15: Offset
8	Time to live (TTL)
9	Type
10 to 11	Checksum
12 to 15	Source IP address
16 to 19	Destination IP address
20 to IHL*4-1	Options + padding to round up to four bytes
Data begins at IHL*4 and ends at Length-1	

Version will for the mean time be 4, although *IP Next Generation* (version 6) is in the process of development. **IHL** is the length of the header divided by 4. **TOS** (*Type of Service*) is a somewhat esoteric field for tuning performance and will not be explained. The **Length** field is the length in bytes of the entire packet inclusive of the header. The **Source** and **Destination** are the IP addresses *from* and *to* where the packet is coming/going. The other fields will be explained soon.

The above description constitutes the view of the Internet that a machine has. However, physically, the Internet consists of many small high speed networks (like a company or a university) called *Local Area Networks*, or *LAN*s. These are all connected to each other via lower speed long distance links. On a LAN, the **raw** medium of transmission is not a packet but an Ethernet *frame*. Frames are analogous to packets (having both a header and a data portion) but are sized to be efficient with particular hardware. IP packets are encapsulated within frames, where the IP packet fits within the **Data** part of the frame. A frame may however be to small to hold an entire IP packet, in which case the IP packet is split into several smaller packets. This group of smaller IP

packets is then given an identifying number and each smaller packet will then have the **Identification** field set with that number and the **Offset** field set to indicate its position within the actual packet. On the other side, the destination machine will reconstruct a packet from all the smaller sub-packets that have the same **Identification** field.

The convention for writing IP address in human readable form in *dotted decimal* notation like 152.2.254.81, where each number is a byte and is hence in the range of 0 to 255. Hence the entire address *space* is in the range of 0.0.0.0 to 255.255.255.255. Now to further organise the assignment of addresses, each 32 bit address is divided into two parts, a *network* and a *host* part of the address.

The network part of the address designates the LAN and the host part the particular machine on the LAN. Now, because it was unknown at the time of specification whether there would one day be more LANs or more machines on a LAN, three different classes of address were created. *Class A* addresses begin with the first bit of the host part set to 0 (hence a Class A address always has the first dotted decimal number less than 128). The next 7 bits give the identity of the LAN and the remaining 24 bits give the identity of an actual machine on that LAN. A Class B address begins with a 1 then a 0 (first decimal number is 128 through 192). The next 14 bits give the LAN and the remaining 16 bits give the machine — most universities, like the address above, are Class B addresses. Finally, Class C addresses start with a 1 1 0 (first decimal number is 192 through 223), and the next 21 bits and then the next 8 bits are the LAN and machine respectively. Small companies tend use Class C addresses.

In practice, there are few organisations that require Class A addresses. A university or large company might use a Class B address, but then it would have its own further subdivisions, like using the third dotted decimal as a department (bits 16 through 23) and the last dotted decimal (bits 24 through 31) as the machine within that department. In this way the LAN becomes a micro Internet in itself. Here the LAN is called a *network* and the various departments are each called a *subnet*.

20.2 Special IP Addresses

There are also some IP addresses that have special purposes that are never used on the open Internet. 192.168.0.0–192.168.255.255 are private addresses perhaps used inside a local LAN that does not communicate directly with the Internet. 127.0.0.0–127.255.255.255 are used for communication with the *localhost* — i.e. the machine itself. Usually 127.0.0.1 is an IP address pointing to the machine itself. 10.0.0.0–10.255.255.255 are additional private address.

20.3 Network Masks and Addresses

Consider again the example of a University with a Class B address. It might have an IP address range of the 137.158.0.0–137.158.255.255. It has decided that the astronomy department should get 512 of its own IP addresses 137.158.26.0–137.158.27.255. We say that astronomy has a *network address* of 137.158.26.0. The machines there all have a *network mask* of 255.255.254.0. A particular machine in astronomy may have an *IP address* of 137.158.27.158. This terminology will be used later.

	Dotted IP	Binary
Netmask	255 . 255 . 254 . 0	1111 1111 1111 1111 1111 1110 0000 0000
Network address	137 . 158 . 26 . 0	1000 1001 1001 1110 0001 1010 0000 0000
IP address	137 . 158 . 27 . 158	1000 1001 1001 1110 0001 1011 1001 1110
Host part	0 . 0 . 1 . 158	0000 0000 0000 0000 0000 0001 1001 1110

20.4 Computers on LAN

Here we will define the term LAN as a network of computers that are all more-or-less connected directly together by Ethernet cables (this is common for the small business with up to about 50 machines). Each machine has an Ethernet card which is refered to as `eth0` when configuring the network from the commandline. If there is more than one card on a single machine, then these are named `eth0`, `eth1`, `eth2` etc. and are each called a *network interface* (or just `interface`) of the machine. LANs work as follows: network cards transmit a frame to the LAN

and other network cards read that frame from the LAN. If any one network card transmits a frame then **all** other network cards can see that frame. If a card starts to transmit a frame while another card is in the process of transmitting a frame, then a *clash* is said to have occurred and the card waits a random amount of time and then tries again. Each network card has a physical address (that is inserted at the time of its manufacture, and has nothing to do with IP addresses) of 48 bits called the *hardware address*. Each frame has a destination address in its header that tells what network card it is destined for, so that network cards ignore frames that are not addressed to them.

Now since frame transmission is governed by the network cards, the destination hardware address must be determined from the destination IP address before sending a packet to a particular machine. The way this is done is through a protocol called the *Address Resolution Protocol* (ARP). A machine will transmit a special packet that asks 'What hardware address is this IP address?'. The guilty machine then responds and the transmitting machine stores the result for future reference. Of course if you suddenly switch network cards, then other machines on the LAN will have the wrong information, so ARP has timeouts and re-requests built into the protocol.

20.5 Configuring Interfaces

Most distributions have a generic way to configure your interfaces. Here we will show the raw method.

We first have to create a `lo` interface. This is called the *loopback* device (and has nothing to do with loopback block devices: `/dev/loop?` files). This is an imaginary device that is used to communicate with the machine itself, if for instance you are `telnet`ing to the local machine, you are actually connecting via the loopback device. The `ifconfig` (*interfaceconfigure*) command is used to do anything with interfaces. First run,

```
/sbin/ifconfig lo down
/sbin/ifconfig eth0 down
```

to delete any existing interfaces, then

```
/sbin/ifconfig lo 127.0.0.1
```

which creates the loopback interface.

The Ethernet interface can be created with:

```
/sbin/ifconfig eth0 192.168.3.9 broadcast 192.168.3.255 netmask 255.255.255.0
```

Now do

```
/sbin/ifconfig
```

to view the interfaces. The output will be,

```
eth0      Link encap:Ethernet  HWaddr 00:00:E8:3B:2D:A2
          inet addr:192.168.3.9  Bcast:192.168.3.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1359 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1356 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:11 Base address:0xe400

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:53175 errors:0 dropped:0 overruns:0 frame:0
          TX packets:53175 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
```

which shows various interesting bits, like the 48 bit hardware address of the network card (`00:00:E8:3B:2D:A2`).

20.6 Configuring Routing

The interfaces are now active, however there is nothing telling the kernel what packets should go to what interface, even though we might expect such behaviour to happen on its own. With UNIX, you must explicitly tell the kernel to send particular packets to particular interfaces.

Any packet arriving through any interface is pooled by the kernel. The kernel then looks at each packet's destination address and decides based on the destination where it should be sent. It doesn't matter where the packet came from, once the kernel *has* it, its what its destination address says that matters. Its up to the rest of the network to ensure that packets do not arrive at the wrong interfaces in the first place.

We know that any packet having the network address `127.???.???.???` must go to the loopback device (this is more or less a convention. The command,

```
/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo
```

adds a *route* to the network `127.0.0.0` albeit an imaginary one.

The `eth0` device can be routed as follows:

```
/sbin/route add -net 192.168.3.0 netmask 255.255.255.0 eth0
```

The command to display the current routes is:

```
/sbin/route -n
```

(`-n` causes `route` to not print IP addresses as hostnames) gives the output

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
192.168.3.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
```

This has the meaning: "packets with destination address `127.0.0.0/255.0.0.0`[1] must be sent to the `lo`opback device", and "packets with destination address `192.168.3.0/255.255.255.0` must be sent to the `eth0`" `Gateway` is zero, hence is not set (see later).

The routing table now routes `127.` and `192.168.3.` packets. Now we need a route for the remaining possible IP addresses. UNIXcan have a route that says to send packets with particular destination IP addresses to another machine on the LAN, from where they might be forwarded elsewhere. This is sometimes called the *gateway* machine. The command is:

```
/sbin/route add -net <network-address> netmask <netmask> gw <gateway-ip-address> <interface>
```

This is the most general form of the command, but its often easier to just type:

```
/sbin/route add default gw <gateway-ip-address> <interface>
```

when we want to add a route that applies to all packets. The `default` signifies all packets; it is the same as

```
/sbin/route add -net 0.0.0.0 netmask 0.0.0.0 gw <gateway-ip-address> <interface>
```

but since routes are ordered according to `netmask`, more specific routes are used in preference to less specific ones.

Finally, you can set your hostname with:

```
hostname cericon.obsidian.co.za
```

A summary of the example commands so far:

[1] The notation *network/mask* is often used to denote ranges of IP address.

```
/sbin/ifconfig lo down
/sbin/ifconfig eth0 down
/sbin/ifconfig lo 127.0.0.1
/sbin/ifconfig eth0 192.168.3.9 broadcast 192.168.3.255 netmask 255.255.255.0
/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo
/sbin/route add -net 192.168.3.0 netmask 255.255.255.0 eth0
/sbin/route add default gw 192.168.3.254 eth0
hostname cericon.obsidian.co.za
```

Although these 7 commands will get your network working, you should not do such a manual configuration. The next section explains how to configure your startup scripts.

20.7 Setting startup scripts

Most distributions will have an modular and extensible system of startup scripts which initiate networking. RedHat systems contain the directory `/etc/sysconfig/`, which contains configuration files to bring up networking automatically.

The file `/etc/sysconfig/network-scripts/` contains:

```
DEVICE=eth0
IPADDR=192.168.3.9
NETMASK=255.255.255.0
NETWORK=192.168.3.0
BROADCAST=192.168.3.255
ONBOOT=yes
```

The file `/etc/sysconfig/network` contains:

```
NETWORKING=yes
FORWARD_IPV4=false
HOSTNAME=cericon.obsidian.co.za
DOMAINNAME=obsidian.co.za
GATEWAY=192.168.3.254
```

You can see that these two files are equivalent to the example configuration done above. There are an enormous amount of options that these two files can take for the various protocols besides TCP/IP, but this is the most common configuration.

The file `/etc/sysconfig/network-scripts/ifcfg-lo` for the loopback device will be configured automatically at installation, you should never need to edit it.

To stop and and start networking (i.e. bring up and down the interfaces and routing), type

```
/etc/rc.d/init.d/network stop
/etc/rc.d/init.d/network start
```

which indirectly will read your `/etc/sysconfig/` files.

20.8 Diagnostic utilities

20.8.1 ping

The `ping` command is the most common network utility. IP packets come in three types on the Internet, represented in the **Type** field of the IP header: *UDP*, *TCP* and *ICMP*. (The former two will be discussed later, and represent the two basic methods of communication between to programs running on different machines.) *ICMP* however, stands for and are diagnostic packets that are responded to in a special way. Try:

```
ping metalab.unc.edu
```

or some other well known host. You will get output like:

```
PING metalab.unc.edu (152.19.254.81) from 192.168.3.9 : 56(84) bytes of data.
64 bytes from 152.19.254.81: icmp_seq=0 ttl=238 time=1059.1 ms
64 bytes from 152.19.254.81: icmp_seq=1 ttl=238 time=764.9 ms
64 bytes from 152.19.254.81: icmp_seq=2 ttl=238 time=858.8 ms
64 bytes from 152.19.254.81: icmp_seq=3 ttl=238 time=1179.9 ms
64 bytes from 152.19.254.81: icmp_seq=4 ttl=238 time=986.6 ms
64 bytes from 152.19.254.81: icmp_seq=5 ttl=238 time=1274.3 ms
64 bytes from 152.19.254.81: icmp_seq=6 ttl=238 time=930.7 ms
```

What is happening is that `ping` is sending ICMP packets to `metalab.unc.edu` which is automatically responding with a return ICMP packet. Being able to `ping` a machine is often the acid test of whether you have communications with it. Note that some site specifically filter ICMP packets, hence `ping cnn.com` doesn't work.

`ping` sends a packet every second and measures the time it takes to receive the return packet — like a submarine sonar "ping". Over the Internet, you can get times in excess of 2 seconds if the place is remote enough. On a local LAN this will drop to under a millisecond.

If ping does not even get to the line `PING metalab.unc.edu`..., it means that it cannot resolve the hostname. You should then check that your DNS is set up correctly — see Chapter 21. If it gets to that line, but no further, it means that the packets are not getting there, or are not getting back. In all other cases, `ping` gives an error message indicating either the absence of routes or interfaces.

20.8.2 `traceroute`

`traceroute` is a rather fascinating utility to identify where a packet has been. It makes use of facilities built into the the ICMP protocol. On my machine,

```
traceroute metalab.unc.edu
```

gives,

```
traceroute to metalab.unc.edu (152.19.254.81), 30 hops max, 38 byte packets
 1  192.168.3.254 (192.168.3.254)  1.197 ms  1.085 ms  1.050 ms
 2  192.168.254.5 (192.168.254.5)  45.165 ms  45.314 ms  45.164 ms
 3  obsgate (192.168.2.254)  48.205 ms  48.170 ms  48.074 ms
 4  obsposix (160.124.182.254)  46.117 ms  46.064 ms  45.999 ms
 5  cismpjhb.posix.co.za (160.124.255.193)  451.886 ms  71.549 ms  173.321 ms
 6  cisap1.posix.co.za (160.124.112.1)  274.834 ms  147.251 ms  400.654 ms
 7  saix.posix.co.za (160.124.255.6)  187.402 ms  325.030 ms  628.576 ms
 8  ndf-core1.gt.saix.net (196.25.253.1)  252.558 ms  186.256 ms  255.805 ms
 9  ny-core.saix.net (196.25.0.238)  497.273 ms  454.531 ms  639.795 ms
10  bordercore6-serial5-0-0-26.WestOrange.cw.net (166.48.144.105)  595.755 ms  595.174 ms *
11  corerouter1.WestOrange.cw.net (204.70.9.138)  490.845 ms  698.483 ms  1029.369 ms
12  core6.Washington.cw.net (204.70.4.113)  580.971 ms  893.481 ms  730.608 ms
13  204.70.10.182 (204.70.10.182)  644.070 ms  726.363 ms  639.942 ms
14  mae-brdr-01.inet.qwest.net (205.171.4.201)  767.783 ms * *
15  * * *
16  * wdc-core-03.inet.qwest.net (205.171.24.69)  779.546 ms  898.371 ms
17  atl-core-02.inet.qwest.net (205.171.5.243)  894.553 ms  689.472 ms *
18  atl-edge-05.inet.qwest.net (205.171.21.54)  735.810 ms  784.461 ms  789.592 ms
19  * * *
20  * * unc-gw.ncren.net (128.109.190.2)  889.257 ms
21  unc-gw.ncren.net (128.109.190.2)  646.569 ms  780.000 ms *
22  * helios.oit.unc.edu (152.2.22.3)  600.558 ms  839.135 ms
```

So you can see that there were twenty machines (or *hops*) between mine and `metalab.unc.edu`.

20.8.3 `tcpdump`

`tcpdump` watches a particular interface for *all* the traffic that passes it — i.e. all the traffic of all the machines connected to the same hub. A network card usually grabs only the frames destined for it, but `tcpdump` puts the card into *promiscuous* mode, meaning for it to retrieve all frames regardless of their destination hardware address. Try

```
tcpdump -n -N -f -i eth0
```

`tcpdump` is also discussed in Section 23.4. Deciphering the output of `tcpdump` is left for now as an exercise for the reader. More on the *tcp* part of `tcpdump` in Chapter **??**.

Chapter 21

DNS and Name Resolution

We know that each computer on the Internet has its own IP address. Although this is sufficient to identify a computer for purposes of transmitting packets, it is not particularly accommodating to people. Also, if a computer were to be relocated we would like to still identify it by the same name.

Hence each computer is given a descriptive textual name. The basic textual name of a machine is called the *unqualified-hostname*[1] and is usually less than eight characters and contains only lowercase letters and numbers (and especially no dots). Groups of computers have a *domainname*. The full name of machine is *unqualified-hostname*.*domainname* and is called the *fully qualified hostname*[2] or the *qualified-hostname*[3] For example, my computer is `cericon`. The domainname of my company is `obsidian.co.za`, and hence the qualified-hostname of my computer is `cericon.obsidian.co.za`, although the IP address might be `160.124.182.1`.

Often the word *domain* usally is synonymous with *domainname*, and the word *hostname* on its own can mean either the qualified or unqualified hostname.

This system of naming computers is called the *Domain Name System (DNS)*

21.1 Top Level Domains (TLD's)

Domain's always end in a standard set of things. Here is a complete list of things that the last bit of a domain can be:

`.com` A US or international `com`pany proper. In fact, any organisation might have a `.com` domain.

`.gov` A US `gov`ernment organisation.

`.edu` A US university.

`.mil` A US `mil`itary department.

`.int` An organisation established by international treaties.

`.org` A US or non-profit `org`anisation. In fact, anyone can have a `.org` domain.

`.net` An Internet service providor. In fact, any bandwidth reseller, IT company or any company at all might have a `.net` domain.

Besides the above, the domain could end in a two letter country code.

The complete list of country codes follows is given in Table 21.1. The `.us` domain is rarely used, since in the US `.com`, `.edu`, `.org`, `.mil`, `.gov`, `.int`, or `.net` are mostly used.

Within each country, a domain may have things before it for better description. Each country may implement a different structure. Some examples are:

[1] This is my own terminology.
[2] Standard terminology
[3] My terminology.

.af Afghanistan	.do Eominican Rep.	.li Liechtenstein	.ws Samoa
.al Albania	.tp East Timor	.lt Lithuania	.sm San Marino
.dz Algeria	.ec Ecuador	.lu Muxembourg	.st Sao Tome and Principe
.as American samoa	.eg Egypt	.mo Macau	.sa Saudi Arabia
.ad Andorra	.sv El Salvador	.mg Madagascar	.sn Senegal
.ao Angola	.gq Equatorial Guinea	.mw Malawi	.sc Seychelles
.ai Anguilla	.ee Estonia	.my Malaysia	.sl Sierra Leone
.aq Antarctica	.et Fthiopia	.mv Maldives	.sg Singapore
.ag Antigua and barbuda	.fk Falkland Islands (Malvinas)	.ml Mali	.sk Slovakia
.ar Argentina	.fo Faroe Islands	.mt Malta	.si Slovenia
.am Armenia	.fj Fiji	.mh Marshall Islands	.sb Solomon Islands
.aw Aruba	.fi Finland	.mq Martinique	.so Somalia
.au Australia	.fr France	.mr Mauritania	.za South Africa
.at Austria	.gf French Guiana	.mu Mauritius	.es Spain
.az Bzerbaijan	.pf French Polynesia	.mx Mexico	.lk Sri Lanka
.bs Bahamas	.tf Grench Southern Territories	.fm Micronesia	.sd Sudan
.bh Bahrain	.ga Gabon	.md Moldova, Rep. of	.sr Suriname
.bd Bangladesh	.gm Gambia	.mc Monaco	.sj Svalbard and Jan Mayen Is.
.bb Barbados	.ge Georgia	.mn Mongolia	.sz Swaziland
.be Belgium	.de Germany	.ms Montserrat	.se Sweden
.bz Belize	.gh Ghana	.ma Morocco	.ch Switzerland
.bj Benin	.gi Gibraltar	.mz Mozambique	.sy Tyrian Arab Rep.
.bm Bermuda	.gr Greece	.mm Nyanmar	.tw Taiwan, Province of China
.bt Bhutan	.gl Greenland	.na Namibia	.tj Tajikistan
.bo Bolivia	.gd Grenada	.nr Nauru	.tz Tanzania, United Rep. of
.ba Bosnia Hercegovina	.gp Guadeloupe	.np Nepal	.th Thailand
.bw Botswana	.gu Guam	.nl Netherlands	.tg Togo
.bv Bouvet Island	.gt Guatemala	.an Netherlands Antilles	.tk Tokelau
.br Brazil	.gn Guinea	.nt Neutral Zone	.to Tonga
.io British Indian Ocean Territory	.gw Guinea-Bissau	.nc New Caledonia	.tt Trinidad and Tobago
.bn Brunei Darussalam	.gy Huyana	.nz New Zealand	.tn Tunisia
.bg Bulgaria	.ht Haiti	.ni Nicaragua	.tr Turkey
.bf Burkina Faso	.hm Heard and Mc Donald Islands	.ne Niger	.tm Turkmenistan
.bi Burundi	.hn Honduras	.ng Nigeria	.tc Turks and Caicos Islands
.by Celarus	.hk Hong Kong	.nu Niue	.tv Uuvalu
.kh Cambodia	.hu Iungary	.nf Norfolk Island	.ug Uganda
.cm Cameroon	.is Iceland	.mp Northern Mariana Islands	.ua Ukraine
.ca Canada	.in India	.no Oorway	.ae United Arab Emirates
.cv Cape Verde	.id Indonesia	.om Pman	.gb United Kingdom
.ky Cayman Islands	.ir Iran (Islamic Rep. of)	.pk Pakistan	.us United States
.cf Central African Rep.	.iq Iraq	.pw Palau	.um US Minor Outlying Islands
.td Chad	.ie Ireland	.pa Panama	.uy Uruguay
.cl Chile	.il Israel	.pg Papua New Guinea	.su USSR
.cn China	.it Jtaly	.py Paraguay	.uz Vzbekistan
.cx Christmas Island	.jm Jamaica	.pe Peru	.vu Vanuatu
.cc Cocos (Keeling) Islands	.jp Japan	.ph Philippines	.va Vatican City State (Holy See)
.co Colombia	.jo Kordan	.pn Pitcairn	.ve Venezuela
.km Comoros	.kz Kazakhstan	.pl Poland	.vn Viet Nam
.cg Congo	.ke Kenya	.pt Portugal	.vg Virgin Islands (British)
.ck Cook Islands	.ki Kiribati	.pr Querto Rico	.vi Wirgin Islands (U.S.)
.cr Costa Rica	.kp Korea, Demo. People's Rep.of	.qa Ratar	.wf Wallis and Futuna Islands
.ci Cote D'ivoire	.kr Korea, Rep. of	.re Reunion	.eh Yestern Sahara
.hr Croatia	.kw Kuwait	.ro Romania	.ye Yemen, Rep. of
.cu Cuba	.kg Lyrgyzstan	.ru Russian Federation	.yu Zugoslavia
.cy Cyprus	.la Lao People's Demo. Rep.	.rw Swanda	.zr Zaire
.cz Czech Rep.	.lv Latvia	.sh St. Helena	.zm Zambia
.cs Dzechoslovakia	.lb Lebanon	.kn Saint Kitts and Nevis	.zw Zimbabwe
.dk Denmark	.ls Lesotho	.lc Saint Lucia	
.dj Djibouti	.lr Liberia	.pm St. Pierre and Miquelon	
.dm Dominica	.ly Libyan Arab Jamahiriya	.vc St. Vincent and the Grenadines	

Table 21.1: ISO country codes

`.co.za` A South African company. (`za` = Zuid Afrika, for the old Dutch postal codes.)

`.org.za` A South African non-profit organisation.

`.ac.za` A South African academic university.

`.edu.au` An australian tertiary educational institution.

`.gov.za` A South African government organisation.

Note that a South African company might choose a `.com` domain or a `.co.za` domain. In our case we use `obsidian.co.za`. The same applies everywhere, so there is no hard and fast rule to locate an organisation from its domain.

21.2 Resolving DNS names to IP addresses

In practice, a user will type a hostname (say `www.obsidian.co.za`) into some application like a web browser. The application has to then try find the IP address associated with that name, in order to send packets to it. This section describes the query structure used on the Internet so that everyone can find out anyone else's IP address.

An obvious way to do this is to distribute a long table of hostname vs. IP numbers to every machine on the Internet. But as soon as you have more than a few thousand machines, this becomes impossible.

Another obvious way to do this is to have one huge computer on the Internet somewhere who's IP address is known by everyone. This computer would be responsible for servicing requests for IP numbers, and the said

application running on your local machine would just query this big machine. Of course with their being billions of machines out their, this will obviously create far too much network traffic[4].

The DNS structure on the Internet actually works like this:

There *are* computers that service requests for IP numbers — millions of them. They are called *name servers* (or *DNS servers*), and a request is called a DNS *lookup*. However, each name server only has information about a specific part of the Internet, and they constantly query each other.

There are 13 *root* name servers on the Internet[5]:

```
a.root-servers.net   198.41.0.4
b.root-servers.net   128.9.0.107
c.root-servers.net   192.33.4.12
d.root-servers.net   128.8.10.90
e.root-servers.net   192.203.230.10
f.root-servers.net   192.5.5.241
g.root-servers.net   192.112.36.4
h.root-servers.net   128.63.2.53
i.root-servers.net   192.36.148.17
j.root-servers.net   198.41.0.10
k.root-servers.net   193.0.14.129
l.root-servers.net   198.32.64.12
m.root-servers.net   202.12.27.33
```

Each country also has a name server, and in turn each organisation has a name server. Each name server only has information about machines in its own domain, as well as information about other name servers. The root name servers only have information on the IP addresses of the name servers of `.com`, `.edu`, `.za` etc. The `.za` name server only has information on the IP addresses of the name servers of `.org.za`, `.ac.za`, `.co.za` etc. The `.co.za` name server only has information on the name servers of all South African companies, like `.obsidian.co.za`, `.icon.co.za`, `.mweb.co.za`, etc. The `.obsidian.co.za`, name server only has info on the machines at Obsidian Systems, like `www.obsidian.co.za`.

Your own machine will have a name server defined in its configuration files that is geographically close to it. The responsibility of this name server will be to directly answer any queries about its own domain that it has information about, and also to answer any other queries by querying as many other name servers on the Internet as is necessary.

Now our application is presented with `www.obsidian.co.za`. The following sequence of lookups take place to resolve this name into an IP address. This procedure is called *hostname resolution* and the algorithm that performs this operation is called the *resolver*.

1. The application will check certain special databases on the local machine. If it can get an answer directly from these, it proceeds no further.

2. The application will look up a geographically close name server from the local machines configuration file. Lets say this machine is called `ns`.

3. The application will query `ns` with "`www.obsidian.co.za`?".

4. `ns` will decide if that IP has been recently looked up before. If it has, there is no need to ask further, since the result would be stored in a local cache.

5. `ns` will see if the domain is local. I.e. if it is a computer that it has direct information about. In this case this would only be true if the `ns` were Obsidian's very own name server.

6. `ns` will strip out the TLD (Top Level Domain) `.za` It will query a root name server, asking what name server is responsible for `.za` The answer will be `ucthpx.uct.ac.za` of IP address `137.158.128.1`.

7. `ns` will strip out the next highest domain `co.za` It will query `137.158.128.1`, asking what name server is responsible for `co.za` The answer will be `secdns1.posix.co.za` of IP address `160.124.112.10`.

[4] Actually, A Microsoft LAN kind of works this way — i.e. not very well

[5] This list can be gotten from `ftp://ftp.rs.internic.net/domain/named.root`

8. ns will strip out the next highest domain obsidian.co.za It will query 160.124.112.10, asking what name server is responsible for obsidian.co.za The answer will be lava.obsidian.co.za of IP address 196.28.133.1.
9. ns will query 196.28.133.1 asking what the IP address is of www.obsidian.co.za The answer will be 160.124.182.1.
10. ns will return the result to the application.
11. ns will store each of these results in a local cache with an expiry date. To avoid having to look them up a second time.

21.3 Configuring your local machine

We made reference to "configuration files" above. These are actually the files: /etc/host.conf, /etc/hosts, and /etc/resolv.conf. These are the three and only files that specify how applications are going to lookup IP numbers, and have nothing to do with the configuration files of the nameserver daemon itself, even thought a nameserver daemon might be running on the local machine.

When an application requires to lookup a hostname it goes through the following procedure[6]. The following are equivalent steps 1, 2 and 3 above with the details of the configuration files filled in. The configuration files that follow are taken as they my might be on my own personal machine:

1. The application will check the file /etc/host.conf. This file will usually have a line order hosts,bind in it, specifying that it should first (hosts) check the local database file /etc/hosts, and then (bind) query the name server specified in /etc/resolv.conf. The file /etc/hosts contains a text readable list of IP addresses and names. An example is given below. If it can get an answer directly from /etc/hosts, it proceeds no further.
2. The application will check in the file /etc/resolv.conf for a line nameserver <nameserver>. There can actually be three of these lines, so that if one nameserver fails, the application can try the next in turn.
3. The application will query that name server with the hostname. If the hostname is unqualified, then it appends the domainname of the local machine to it before trying the query. If the keyword search <domain1> <domain2> ... <domainN> appears in the configuration file, then it tries a query with each of <domain1>, <domain2> etc. appended in turn until the query successfully returns an IP. This just saves you having to type in the full hostname for computers within your own organisation.
4. The nameserver will proceed with the hierarchical queries described from step 4 onward.

The /etc/hosts file should look something like this:

```
127.0.0.1           localhost.localdomain          localhost
192.168.3.9         cericon.obsidian.co.za         cericon
192.168.3.10        aragorn.obsidian.co.za         aragorn
192.168.2.1         ra.obsidian.co.za              ra
```

The hosts aragorn, cericon and ra are the hosts I am most interested in, and hence are listed here. cericon is my local machine and must be listed. You can list any hosts that you want fast lookups too, or hosts that might need to be known in spite of nameservers being down.

The /etc/host.conf might look like this. All of the lines are optional:

```
order     hosts, bind, nis
trim      some.domain
spoofalert
nospoof
multi     on
reorder
```

[6] What is actually happening is that the application is making a C library call to the function gethostbyname(), hence all these configuration files really belong to the C library packages glibc or libc. However this is a detail you need not be concerned about.

`order` The order in which lookups are done. Don't try fiddling with this value. It never seems to have any effect. You should leave it as `order hosts,bind` (or `order hosts,bind,nis` if you are using NIS – discussed in later chapters). Once again, `bind` means to then go and check the `/etc/resolv.conf` which holds the nameserver query options.

`trim` Strip the domain `some.domain` from the end of a hostname before trying a lookup. You will probably never require this feature.

`spoofalert` Try reverse lookups on a hostname after looking up the IP (i.e. do a query to find the name from the IP.) If this query does not return the correct result, it could mean that someone is trying to make it look like they are a machine that they aren't. This is a hackers trick called *spoofing*. This warns you of such attempts in your log file `/var/log/messages` (see Section **??**).

`nospoof` Disallows results that fail this spoof test.

`multi on` Return more than one result if there are aliases. Actually, a host can have several IP numbers and an IP number can have several hostnames. Consider a computer that might want more than one name (`ftp.obsidian.co.za` and `www.obsidian.co.za` are the same machine.) Or a machine that has several networking cards and an IP address for each. This should always be turned on. `multi off` is the alternative. Most applications use only the first value returned.

`reorder` Reorder says that if more than one IP is returned by a lookup, then that list should be sorted according to the IP that has the most convenient network route.

Despite this array of options, an `/etc/host.conf` file almost always looks simply like:

```
order     hosts, bind
multi     on
```

The `/etc/resolv.conf` file could look something like this:

```
nameserver 192.168.2.1
nameserver 160.124.182.1
nameserver 196.41.0.131
search obsidian.co.za ct.obsidian.co.za uct.ac.za
sortlist 192.168.3.0/255.255.255.0 192.168.2.0/255.255.255.0
options ndots:1 timeout:30 attempts:2 rotate no-check-names inet6
```

`nameserver` Specifies a nameserver to query. No more than three may be listed. The point of having more than one is to safeguard against a nameserver being down — the next in the list will then just be queried.

`search` If given a hostname with less than `ndots` dots (i.e. `1` in this case), add each of the domains in turn to the hostname, trying a lookup with each. This allows you to type in an unqualified hostname and have application work out what organisation it is belongs to from the search list. You can have up to six domains, but then queries would be a time-consuming.

`domain` The line "`domain ct.obsidian.co.za`" is the same as "`search ct.obsidian.co.za obsidian.co.za co.za`". Always use `search` explicitly instead of `domain` to reduce the number of queries to a minimum.

`sortlist` If more than one host is returned, sort them according to the following *network/masks*.

`options` Various additional parameters can be specified in this one line:

- `ndots` Explained under `search` above. The default is one.
- `timeout` How long to wait before considering a query to have failed. The default is 30 seconds.
- `attempts` Number of attempts to make before failing. This defaults to 2. This means that a down nameserver will cause your application to wait 1 full minute before deciding that it can't resolve the IP.
- `rotate` Try the nameservers at in round robin fashion. This distributes load across nameservers.
- `no-check-names` Don't check for invalid characters in hostnames.

inet6 The man page for resolv.conf (resolver(5)) says:

```
inet6      sets RES_USE_INET6 in _res.options .  This has the ef-
           fect of trying a AAAA query before an A query inside
           the gethostbyname function, and of mapping IPv4 re-
           sponses in IPv6 ‘‘tunnelled form’’ if no AAAA records
           are found but an A record set exists.
```

But I don't know what an AAAA query is.

Despite this array of options, an /etc/resolv.conf file almost always looks simply like:

```
nameserver 192.168.2.254
search obsidian.co.za
```

21.4 Reverse lookups

A *reverse lookup* was mentioned under nospoof above. This is the determining of the hostname from the IP address. The course of queries is similar to forward lookups using part of the IP address to find out what machines are responsible for what ranges of IP address.

A *forward lookup* is an ordinary lookup of the IP address from the hostname.

21.5 *Authoritive* for a domain

It has been emphasised that name servers only hold information for their own domains. Any other information they may have about another domain is cached, temporary data that has an expiry date attached to it.

The domain that a name server has information about is said to be the domain that a name server is *authoritive* for. Alternatively we say: "a name server is *authorative* for the domain". For instance, the server ns2.obsidian.co.za is authoritive for the domain obsidian.co.za. Hence lookups from anywhere on the Internet having the domain obsidian.co.za ultimately are the responsibility of ns2.obsidian.co.za, and originate (albeit via a long series of caches) from the host ns2.obsidian.co.za.

21.6 The host, ping and whois command

The command host looks up a hostname or an IP address. Try:

```
host www.cnn.com
```

for an example of a host with lots of IP address. Keep typing host over and over. Notice that the order of the hosts keeps changing randomly. This is to distribute load amidst the many cnn.com servers.

Now pick one of the IP addresses and type

```
host <ip-address>
```

This will return the hostname cnn.com.

Note that the host command is not available on all UNIX's

The ping command has nothing directly to do with DNS, but is a quick way of getting an IP address, and checking if a host is responding at the same time. It is often used as the acid test for network and DNS connectivity. See Section 20.8.1.

Now enter:

```
whois cnn.com@rs.internic.net
```

(Note that original BSD whois worked like whois -h <host> <user>) You will get a response like:

```
[rs.internic.net]

Whois Server Version 1.1

Domain names in the .com, .net, and .org domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: CNN.COM
   Registrar: NETWORK SOLUTIONS, INC.
   Whois Server: whois.networksolutions.com
   Referral URL: www.networksolutions.com
   Name Server: NS-01A.ANS.NET
   Name Server: NS-01B.ANS.NET
   Name Server: NS-02A.ANS.NET
   Name Server: NS-02B.ANS.NET
   Updated Date: 22-sep-1999


>>> Last update of whois database: Thu, 20 Jan 00 01:39:07 EST <<<

The Registry database contains ONLY .COM, .NET, .ORG, .EDU domains and
Registrars.
```

[7]

(Internic happens to have this database of .com, .net, .org and .edu domains.)

21.7 The nslookup command

nslookup is a program to interactively query a nameserver. If you run

```
nslookup
```

you will get a > prompt where you can type commands. If you type in a hostname, nslookup will return its IP address(s) and visa versa. Also typing

```
help
```

any time will return a complete list of commands. By default, nslookup uses the first nameserver listed in /etc/resolv.conf for all its queries. However, the command

```
server <nameserver>
```

can be used to explicitly use a particular server.

21.7.1 NS, MX, PTR, A and CNAME records

The word *record* is a piece of DNS information.

Now enter the command:

```
set type=NS
```

[7] An example of Y2K non-compliancy

This tells nslookup to return the second type of information that a DNS can deliver: *the authoritive name server for a domain* or the NS record of the domain. You can enter any domain here. For instance, enter

```
set type=NS
cnn.com
```

It will return

```
Non-authoritative answer:
cnn.com nameserver = NS-02B.ANS.NET
cnn.com nameserver = NS-02A.ANS.NET
cnn.com nameserver = NS-01B.ANS.NET
cnn.com nameserver = NS-01A.ANS.NET

Authoritative answers can be found from:
NS-02B.ANS.NET  internet address = 207.24.245.178
NS-02A.ANS.NET  internet address = 207.24.245.179
NS-01B.ANS.NET  internet address = 199.221.47.8
NS-01A.ANS.NET  internet address = 199.221.47.7
```

This tells us that their are four name servers authoritive for the domain cnn.com (one plus three backups). It also tells us that it did not get this answer from an authoritive source, but via a cached source. It also tells us what nameservers are authorative for this very information.

Now switch to a nameserver that *is* authoritive for cnn.com:

```
server NS-02B.ANS.NET
```

and run the same query:

```
cnn.com
```

The new result is somewhat more emphatic, but no different.

There are only a few other kinds of records that you can get from a nameserver. Try

```
set type=MX
cnn.com
```

to get the so-called *MX record* for that domain. The MX record is the server responsible for handling mail destined to that domain. MX records also have a priority (usually 10 or 20). This tells any mail server to try the 20 one should the 10 one fail, and so on. There are usually only one or two MX records. Mail is actually the only Internet service handled by DNS. (For instance there is no such thing as a NEWSX record for news, nor a WX record for web pages, whatever kind of information we may like such records to hold.)

Also try

```
set type=PTR
<ip-address>
set type=A
<hostname>
set type=CNAME
<hostname>
```

So-called PTR records are reverse lookups, or *PoinTeRs* to hostnames. So-called *A* records are forward lookups (the default type of lookup when you first invoke nslookup), or *Address* lookups. So-called CNAME records are lookups of *Canonical NAMEes*. DNS allows one to alias a computer to many different names, even though each has one *real* name (called the *canonical* name). CNAME lookups returns the machine name proper.

Chapter 22

named — Domain Name Server

This chapter follows on from Chapter 21.

There seems to be a lot of hype that elevates the name server to something mystical and illusive. In fact, setting up a nameserver is a standard and trivial exercise.

A nameserver daemon is also no heavyweight service: The named executable is 500kB, and consumes little CPU.

The package that the name server comes in is called bind. This chapter assumes a bind of approximately bind-8.2 or later. bind stands for *Berkeley Internet Name Domain*.

The difficulty with setting up a nameserver is that the configuration files are impossible to construct from a specification without making some kind of typing error.

The solution is quite simple: **never** create a nameserver config file from scratch. **always** copy one from an existing working name server. Here we will give more example configuration files than explanation. You can copy these examples to create your own nameserver.

Before you even start working on nameserver configuration, you should start a new terminal window with the command:

```
tail -f /var/log/messages
```

Keep this window throughout the entire setup and testing procedure. From now on, when I refer to *messages* I am refering to a message in this window.

Documentation

The man page for named are hostname(7), named-xfer(8), named(8), and ndc(8).

The man pages reference a document called the "Name Server Operations Guide for BIND". What they actually mean is a text file /usr/doc/bind-8.2/bog/file.lst or a PostScript file /usr/doc/bind-8.2/bog/file.psf for printing.

The problem with some of this documentation is that it is still based on the old (now depreciated) named.boot configuration file. There is a program /usr/doc/bind-8.2/named-bootconf/named-bootconf that reads a named.boot file from stdin and writes a named.conf file to stdout. I found it useful to echo "old config line" | named-bootconf to see what a new style equivalent would be.

The most important info is in /usr/doc/bind-8.2/html which contains a complete reference to configuration.

There are also FAQ documents in /usr/doc/bind-8.2/misc and various thesis on security. /usr/doc/bind-8.2/misc/style.txt contains the recommended layout of the configuration files for consistent spacing and readability. Finally /usr/doc/bind-8.2/rfc contains the relevant RFC's (See Section ??).

Configuration files

There is only one main configuration file for named: /etc/named.conf. The named service once used a file /etc/named.boot but this has been scrapped. If there is a named.boot file in your /etc directory then it is not being used, except possibly by a very old version of bind.

The named.conf file will have a line in it directory "/var/named"; or directory "/etc/named";. This directory hold various files containing textual lists of name to IP address mappings. The following example is a nameserver for a company that has been given a range of IP address (196.28.133.20–30), as well as one single IP address (160.124.182.44). It also must support a range of internal IP addresses (192.168.2.0–255) The trick is not to think about how everything works. If you just copy and edit things in a consistent fashion, carefully reading the comments, this will work fine.

The /etc/name.conf file should look like:

```
/*
 * The ``directory'' line tells named that any further file name's
 * given are under the /var/named/ directory.
 */
options {
        directory "/var/named";
        /*
         * If there is a firewall between you and nameservers you want
         * to talk to, you might need to uncomment the query-source
         * directive below.  Previous versions of BIND always asked
         * questions using port 53, but BIND 8.1 uses an unprivileged
         * port by default.
         */
        // query-source address * port 53;
};

/* The list of root servers: */
zone "." {
        type hint;
        file "named.ca";
};

/* Forward lookups of hosts in my domain: */
zone "obsidian.co.za" {
        type master;
        file "named.obsidian.co.za";
};

/* Reverse lookups of the localhost: */
zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local";
};

/* Reverse lookups of local IP numbers: */
zone "1.168.192.in-addr.arpa" {
        type master;
        file "named.192.168.1";
};

/* Reverse lookups of 196.28.133.* Internet IP numbers: */
zone "133.28.196.in-addr.arpa" {
        type master;
        file "named.196.28.133";
};

/* Reverse lookup of 160.124.182.44 only: */
zone "44.182.124.160.in-addr.arpa" {
        type master;
```

```
        file "named.160.124.182.44";
};
```

The /var/named.ca file should look like:

```
; Get the original of this file from ftp://ftp.rs.internic.net/domain/named.root
;
; formerly ns.internic.net
.                        3600000  IN  NS    a.root-servers.net.
a.root-servers.net.      3600000      A     198.41.0.4
.                        3600000      NS    b.root-servers.net.
b.root-servers.net.      3600000      A     128.9.0.107
.                        3600000      NS    c.root-servers.net.
c.root-servers.net.      3600000      A     192.33.4.12
.                        3600000      NS    d.root-servers.net.
d.root-servers.net.      3600000      A     128.8.10.90
.                        3600000      NS    e.root-servers.net.
e.root-servers.net.      3600000      A     192.203.230.10
.                        3600000      NS    f.root-servers.net.
f.root-servers.net.      3600000      A     192.5.5.241
.                        3600000      NS    g.root-servers.net.
g.root-servers.net.      3600000      A     192.112.36.4
.                        3600000      NS    h.root-servers.net.
h.root-servers.net.      3600000      A     128.63.2.53
.                        3600000      NS    i.root-servers.net.
i.root-servers.net.      3600000      A     192.36.148.17
.                        3600000      NS    j.root-servers.net.
j.root-servers.net.      3600000      A     198.41.0.10
.                        3600000      NS    k.root-servers.net.
k.root-servers.net.      3600000      A     193.0.14.129
.                        3600000      NS    l.root-servers.net.
l.root-servers.net.      3600000      A     198.32.64.12
.                        3600000      NS    m.root-servers.net.
m.root-servers.net.      3600000      A     202.12.27.33
```

The /var/named.obsidian.co.za file should look like:

```
@               IN      SOA     ns1.obsidian.co.za. root.ns1.obsidian.co.za. (
                2000012101      ; Serial number
                10800           ; Refresh every 3 hours
                3600            ; Retry every hour
                3600000         ; Expire after 42 days
                259200 )        ; Minimum Time to Live (TTL) of 3 days

                IN      NS      ns1.obsidian.co.za.
                IN      NS      ns2.obsidian.co.za.

                IN      A       160.124.182.44
                IN      MX      10 mail1.obsidian.co.za.
                IN      MX      20 mail2.obsidian.co.za.

ns1             IN      A       196.28.144.1
ns2             IN      A       196.28.144.2
ftp             IN      A       196.28.133.3

www             IN      CNAME   obsidian.co.za.
mail1           IN      CNAME   ns1.obsidian.co.za.
mail2           IN      CNAME   ns2.obsidian.co.za.
gopher          IN      CNAME   ftp.obsidian.co.za.
pop             IN      CNAME   mail1.obsidian.co.za.
proxy           IN      CNAME   ftp.obsidian.co.za.

http            IN      CNAME   www.obsidian.co.za.
```

```
pc1             IN      A       192.168.2.1
pc2             IN      A       192.168.2.2
pc3             IN      A       192.168.2.3
pc4             IN      A       192.168.2.4
```

The /var/named.local file should look like:

```
@               IN      SOA     localhost. root.localhost. (
                2000012101      ; Serial number
                10800           ; Refresh every 3 hours
                3600            ; Retry every hour
                3600000         ; Expire after 42 days
                259200 )        ; Minimum Time to Live (TTL) of 3 days

                IN      NS      localhost.

1               IN      PTR     localhost.
```

The /var/named.192.168.1 file should look like:

```
@               IN      SOA     localhost. root.localhost. (
                2000012101      ; Serial number
                10800           ; Refresh every 3 hours
                3600            ; Retry every hour
                3600000         ; Expire after 42 days
                259200 )        ; Minimum Time to Live (TTL) of 3 days

                IN      NS      localhost.

1               IN      PTR     pc1.obsidian.co.za.
2               IN      PTR     pc2.obsidian.co.za.
3               IN      PTR     pc3.obsidian.co.za.
4               IN      PTR     pc4.obsidian.co.za.
```

The /var/named.196.28.133 file should look like:

```
@               IN      SOA     ns1.obsidian.co.za. dnsmaster.ns1.obsidian.co.za. (
                2000012101      ; Serial number
                10800           ; Refresh every 3 hours
                3600            ; Retry every hour
                3600000         ; Expire after 42 days
                259200 )        ; Minimum Time to Live (TTL) of 3 days

                IN      NS      ns1.obsidian.co.za.
                IN      NS      ns2.obsidian.co.za.

1               IN      PTR     ns1.obsidian.co.za.
2               IN      PTR     ns2.obsidian.co.za.
3               IN      PTR     ftp.obsidian.co.za.
```

The /var/named.160.124.182.44 file should look like:

```
@               IN      SOA     ns1.obsidian.co.za. dnsmaster.ns1.obsidian.co.za. (
                2000012101      ; Serial number
                10800           ; Refresh every 3 hours
                3600            ; Retry every hour
                3600000         ; Expire after 42 days
                259200 )        ; Minimum Time to Live (TTL) of 3 days

                IN      NS      ns1.obsidian.co.za.
                IN      NS      ns2.obsidian.co.za.

```

```
                IN      PTR     www.obsidian.co.za.
```

Run the appropriate lines:

```
/etc/rc.d/init.d/named start
/etc/rc.d/init.d/named stop
/etc/rc.d/init.d/named restart
```

You should get messages like:

```
Jan 21 13:41:04 ns1 named[24996]: starting.  named 8.2 Fri Jan 20 11:15:20 EST 1999 ^Iroot@ns1.obsidian.co.za:
    /usr/src/bs/BUILD/bind-8.2/src/bin/named
Jan 21 13:41:04 ns1 named[24996]: cache zone "" (IN) loaded (serial 0)
Jan 21 13:41:04 ns1 named[24996]: Zone "obsidian.co.za" (file named.obsidian.co.za):
    No default TTL set using SOA minimum instead
Jan 21 13:41:04 ns1 named[24996]: master zone "obsidian.co.za" (IN) loaded (serial 2000012101)
Jan 21 13:41:04 ns1 named[24996]: Zone "0.0.127.in-addr.arpa" (file named.local):
    No default TTL set using SOA minimum instead
Jan 21 13:41:04 ns1 named[24996]: master zone "0.0.127.in-addr.arpa" (IN) loaded (serial 2000012101)
Jan 21 13:41:04 ns1 named[24996]: Zone "1.168.192.in-addr.arpa" (file named.192.168.1):
    No default TTL set using SOA minimum instead
Jan 21 13:41:04 ns1 named[24996]: master zone "1.168.192.in-addr.arpa" (IN) loaded (serial 2000012101)
Jan 21 13:41:04 ns1 named[24996]: Zone "133.28.196.in-addr.arpa" (file named.196.28.133):
    No default TTL set using SOA minimum instead
Jan 21 13:41:04 ns1 named[24996]: master zone "133.28.196.in-addr.arpa" (IN) loaded (serial 2000012101)
Jan 21 13:41:04 ns1 named[24996]: Zone "44.182.124.160.in-addr.arpa" (file named.160.124.182.44):
    No default TTL set using SOA minimum instead
Jan 21 13:41:04 ns1 named[24996]: master zone "44.182.124.160.in-addr.arpa" (IN) loaded (serial 2000012101)
Jan 21 13:41:04 ns1 named[24996]: listening on [127.0.0.1].53 (lo)
Jan 21 13:41:04 ns1 named[24996]: listening on [192.168.3.9].53 (eth0)
Jan 21 13:41:04 ns1 named[24996]: Forwarding source address is [0.0.0.0].1060
Jan 21 13:41:04 ns1 named[24997]: Ready to answer queries.
```

If you have made typing errors, or named files incorrectly, you will get appropriate error messages. **Novice administrators are want to edit `named` configuration files and restart `named` without checking `/var/log/messages` for errors**. ***NEVER do this***.

Configuration file details

The top-level configuration file `/etc/named.conf` has an obvious format **C** style format. Comments are designated by `/* */`, `//`, or `#`.

The `options` section in our case specifies only one parameter: the directory for locating any files. `/usr/doc/bind-8.2/html/options.html` has a complete list of options.

The lines `zone "." {`... will be present in almost all nameserver configurations. It tells `named` that the whole Internet is governed by the file `named.ca`. `named.ca` in turn contains the list of root nameservers.

The lines `zone "0.0.127.in-addr.arpa" {`... will also always be present. It specifies that reverse lookups for the IP address range `127.0.0.0`–`255` are stored in the file `named.local`. (Note that `0.0.127` is `127.0.0` written backwards. In fact, reverse lookups are just forward lookups under the domain `.in-addr.arpa`.

The rest of the file is the configuration specific to our domain.

The lines `zone "obsidian.co.za" {`... says that info for forward lookups is located in the file `named.obsidian.co.za`.

The lines `zone "1.168.192.in-addr.arpa" {`... says that info for reverse lookups on the IP address range `192.168.1.0`–`255` is located in the file `named.192.168.1`.

The lines `zone "44.182.124.160.in-addr.arpa" {`... says that info for reverse lookups on the IP address `160.124.182.44` is located in the file `named.160.124.182.44`.

Domain `SOA` records

Each of the above `named.` files has a similar format. They begin with an `@ IN SOA`. `SOA` stands for *Start of Authority*. The hostname on the first line specifies the authority for that domain, and the adjacent `<user>.<hostname>` specifies the email address of the responsible person.

The next few lines contain timeout specifications for cached data and data propagation across the net. These are reasonable defaults, but if you would like to tune these values, consult the relevant documentation listed above. The values are all in seconds.

The *serial number* for the file (i.e. 2000012101) is used to tell when a change has been made and hence that new data should be propagated to other servers. When updating the file in any way, this serial number should be incremented. The format is conventionally *YYYYMMDDxx* — exactly ten digits. *xx* begins with, say, 01 and is incremented with each change made during a day.

It is absolutely essential that the serial number be updated whenever a file is edited. If not, the changes will not be reflected through the rest of the Internet.

Dotted and non-dotted hostnames

If a hostname in a ends in a . then it signifies that it a fully qualified hostname. If it does not end in a . then it signifies that the domain should be appended to the hostname. This feature is purely to make files more elegant.

For instance, The line

```
ftp                     IN      A       196.28.133.3
```

could just as well be written

```
ftp.obsidian.co.za.     IN      A       196.28.133.3
```

Always be careful to properly end qualified hostnames with a dot, since failing to do so causes named to append a further domain.

Empty hostnames

An omitted hostname is substitute with the domain. The purpose of this notation is also for elegance. For example

```
                        IN      NS      ns1.obsidian.co.za.
```

is the same as

```
obsidian.co.za.         IN      NS      ns1.obsidian.co.za.
```

NS, MX, PTR, A and CNAME records

Each DNS record appears on a single line, associating some hostname/domain or IP address with some other hostname or IP address.

It is hence easy to construct a file that makes the Internet think anything you want it to about your organisation.

The most basic type of record is the A and PTR records. They simply associates a hostname with an IP number, or an IP number with a hostname respectively. You should not have more than one host associated to a particular IP number.

The CNAME record says that a host is just an alias to another host. So rather have

```
ns1             IN      A       196.28.144.1
mail1           IN      CNAME   ns1.obsidian.co.za.
```

than,

```
ns1             IN      A       196.28.144.1
mail1           IN      A       196.28.144.1
```

Finally, NS and MX records,

```
<domain>        IN      NS      <nameserver>
<domain>        IN      MX      <mailserver>
```

just state that domain <domain> has a nameserver/mailserver <nameserver> or <mailserver> respectively.

22.1 Configuring named for dialup use

If you have a dialup connection, the nameserver should be configured as what is called a *caching-only* nameserver. Of course their is no such thing as a caching-*only* nameserver — it just means that the name. files have only a few essential records in them. The point of a caching server is to prevent spurious DNS lookups that may eat modem bandwidth or cause a dial-on-demand server to initiate a dialout. It also prevents applications blocking waiting for DNS lookup. (A typical example of this is sendmail, which blocks for couple of minutes when a machine is turned on without the network plugged in; and netscape 4, which tries to look up the IP address of news.<localdomain>.)

The /etc/name.conf file should look as follows. Replace <naneserver> with the IP address of the nameserver your ISP has given you. Your local machine name is assumed to be cericon.priv.ate. (The following listings are minus superfluous comments and newlines for the purposes of brevity):

```
options {
        forwarders {
                <nameserver>;
        };
        directory "/var/named";
};

zone "." { type hint; file "named.ca"; };
zone "priv.ate" { type master; file "named.priv.ate"; };
zone "0.0.127.in-addr.arpa" { type master; file "named.local"; };
zone "168.192.in-addr.arpa" { type master; file "named.192.168"; };
```

The /var/named.ca file is the same as before. The /var/named.priv.ate file should look like:

```
@       IN      SOA     cericon.priv.ate. root.cericon.priv.ate.
        ( 2000012101 10800 3600 3600000 259200 )
        IN      NS      cericon.priv.ate.
cericon IN      A       192.168.1.1
news    IN      A       192.168.1.2
```

The /var/named.local file should look like:

```
@       IN      SOA     localhost. root.localhost.
        ( 2000012101 10800 3600 3600000 259200 )
        IN      NS      localhost.
1       IN      PTR     localhost.
```

The /var/named.192.168 file should look like:

```
@       IN      SOA     localhost. root.localhost.
        ( 2000012101 10800 3600 3600000 259200 )
        IN      NS      localhost.
1.1     IN      PTR     cericon.priv.ate.
```

In addition to the above, your hostname and domainname have to be configured as per Chapter 21.

Dynamic IP addresses

The one contingency of dialup machines is that IP addresses are often dynamically assigned. So your 192.168. addresses aren't going to apply. Probably one way to get around this is to get a feel for what IP addresses you are likely to get by dialling in a few times. Assuming you know that your ISP always gives you 196.26.*x*.*x*, you can have a reverse lookup file named.196.26 with nothing in it. This will just cause reverse lookups to fail instead of blocking.

This is actually a bad idea because an application may legitimately need to reverse lookup in this range. The real complete solution would involve creating a script to modify the named.conf file and restart named upon each dialup.

For instance, pppd (from the ppp-2.*x*.*x* package) executes a user defined script upon a successful dial. This script would be run by pppd after determining the new IP address. The script should create a complete named configuration based on the current IP and then restart named.

In Section 23.3 we show a dynamic DNS configuration that does this.

Both of these plans may be unnecessary. It is probably best to identify the particular application that is causing a spurious dial-out, or causing a block, and then apply your creativity for the particular case. For instance, in my own case, a setup had netscape taking minutes to start up — rather irritating to the user. I immediately diagnosed that netscape was trying to do a reverse lookup of some sort. An strace revealed that it was actually trying to find a news server on the local domain. Simply creating a news record pointing to the local machine fixed the problem[1].

22.2 Secondary or slave DNS servers

named can operate as a backup server to another server also called a *slave* or *secondary* server.

Like the caching-*only* server there is no such thing as a *secondary* server. Its just the same named running with reduced info.

Lets say we would like ns2.obsidian.co.za to be a secondary to ns1.obsidian.co.za. The named.conf file would look as follows:

```
options {
        directory "/var/named";
        // query-source address * port 53;
};

/* The list of root servers: */
zone "." {
        type hint;
        file "named.ca";
};

/* Forward lookups of hosts in my domain: */
zone "obsidian.co.za" {
        type slave;
        file "named.obsidian.co.za";
        masters {
                196.28.144.1;
        };
};

/* Reverse lookups of the localhost: */
zone "0.0.127.in-addr.arpa" {
        type master;
        file "named.local";
};
```

[1] Actually it could also have been fixed in the Netscape configuration were the news server can be specified.

```
/* Reverse lookups of local IP numbers: */
zone "1.168.192.in-addr.arpa" {
        type slave;
        file "named.192.168.1";
        masters {
                196.28.144.1;
        };
};

/* Reverse lookups of 196.28.133.* Internet IP numbers: */
zone "133.28.196.in-addr.arpa" {
        type slave;
        file "named.196.28.133";
        masters {
                196.28.144.1;
        };
};

/* Reverse lookup of 160.124.182.44 only: */
zone "44.182.124.160.in-addr.arpa" {
        type slave;
        file "named.160.124.182.44";
        masters {
                196.28.144.1;
        };
};
```

Where an entry has a "master" in it, you must supply the appropriate file. Where an entry has a "slave" in it, named will automatically download the file from 196.28.144.1 (i.e. ns1.obsidian.co.za) the first time a lookup is required from that domain.

An that's DNS!

Chapter 23

Point to Point Protocol — Dialup Networking

Dial up networking is unreliable and difficult to configure. This is simply because telephones were not designed for data. However, considering that the telephone network is by far the largest electronic network on the globe, it makes sense to make use of it. This is why modems were created. On the other hand the recent advent of ISDN is slightly more expensive and a better choice for all but home dial-up. See Section 23.5 for more info.

23.1 Basic Dialup

For home use, dial up network is not all that difficult to configure. `/usr/doc/HOWTO/PPP-HOWTO` contains lots on this. For my machine this boils down to creating the files `/etc/ppp/chap-secrets` and `/etc/ppp/pap-secrets` both containing the following line of text:

```
<username>  *  <password>  *
```

And then running the following command at a shell prompt:

```
pppd connect \
            "chat -S -s -v \
            '' 'AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0' \
            OK ATDT<tel-number> CONNECT '' \
            name: <username> assword: '\q<password>' \
            con: ppp" \
    /dev/<modem> 57600 debug crtscts modem lock nodetach \
    hide-password defaultroute \
    user <username>
```

This is a minimalists dial in command and it's specific to **my** ISP only. Don't use the exact command unless you have an account with the *Internet Solution* ISP in South Africa.

The command-line options are explained as follows:

`connect <script>` This is the script that `pppd` is going to use to start things up. When you use a modem manually (as you will be shown further below), you need to go through the steps of initialising the modem, causing a dial, connecting, logging in, and finally telling the remote computer that you would like to start modem *data communication* mode, called the *point to point protocol*, or *PPP*. The `<script>` is the automation of this manual procedure.

`chat -S -s -v <expect> <send> <expect> <send> ...` This is the `<script>` proper. `chat` has a man page and other uses besides with modem communications. `-S` means to log messages to the terminal and not to SYSLOG; `-s` means to log to stderr; `-v` means verbose output. After the options, comes a list of things the modem is likely to say, alternated with appropriate responses. This is called an *expect–send* sequence. The `AT S7=...` sequence initialises the modem to something we would like. For many modems this could just be `ATZ`. What works best for your modem can be discovered by looking in the manual that came with

it. It will talk about the AT command set in detail and is essential reading for anyone doing serious PPP work. \q means to **not** print the password amidst the debug output — very important.

/dev/tty?? This tells the device you are going to use. This will usually be /dev/ttyS0, /dev/ttyS1, /dev/ttyS2 or /dev/ttyS3.

57600 The speed the modem is to be set to. This is only the speed between the PC and the modem, and has nothing to do with the actual data throughput. It should be set as high as possible except in the case of very old machines whose serial ports may possibly only handle 38400.

debug is to output debug information. This is useful for diagnosing problems.

crtscts Use hardware flow control.

modem Use modem control lines. This is actually the default.

lock Create a *UUCP style* lock file in /var/lock/. This is just a file of the form /var/lock/LCK..tty?? that tells other applications that the serial device is in use. For this reason, you **must not** call the device /dev/modem or /dev/cua?.

nodetach Don't go into the background. This allows you to watch pppd run and stop it with ^C.

defaultroute Create an IP route after PPP comes alive. Henceforth, packages will go to the right place.

hide-password Do not show the password in the logs. This is important for security.

user <username> Specifies the line from the /etc/ppp/chap-secrets and /etc/ppp/pap-secrets file to use. There is usually only one.

23.1.1 Determining your chat script

To determine the list of expect–send sequences, you need to do a manual dial in. The command

```
dip -t
```

stands for *dial-IP* and talks directly to your modem.

The following session demonstrates a manual dial for user psheer. Using dip manually like this is a game of trying to get the garbage lines you see below: this is PPP starting to talk. When you get this junk you have won, and can press ^C. Then paste your session for future reference.

```
[root@cericon root]# dip -t
DIP: Dialup IP Protocol Driver version 3.3.7o-uri (8 Feb 96)
Written by Fred N. van Kempen, MicroWalt Corporation.

DIP> port ttyS0
DIP> speed 57600
DIP> term
[ Entering TERMINAL mode.  Use CTRL-] to get back ]
ATDT4068500
CONNECT 26400/ARQ/V34/LAPM/V42BIS
Checking authorization, lease wait...
name:psheer
password:

c2-ctn-icon:ppp
Entering PPP mode.
Async interface address is unnumbered (FastEthernet0)
Your IP address is 196.34.157.148.  MTU is 1500 bytes

~y}#A!}!e} }3}"}&} }*} } }~}&4}2 Iq}'}"}(}" N$~~y}#A!}!r} }4}"}&} }
[ Back to LOCAL mode. ]
DIP> quit
You have mail in /var/spool/mail/root
[root@cericon root]#
```

Now you can modify the above chat script as you need. The kinds of things that will differ are trivial: like having login: instead of name:. Some also require you to type something instead of ppp, and some require nothing to be typed after your password.

23.1.2 CHAP and PAP

You may ask why there are `/etc/ppp/chap-secrets` and `/etc/ppp/pap-secrets` files if a username and password is already specified inside the the `chat` script. *CHAP* (Challenge Handshake Authentication Protocol) and *PAP* (Password Authentication Protocol) are authentication mechanisms used *after* logging in — in other words, somewhere amidst the `~y}#A!}!e} }3}"}&} }*} } }~}&4}2Iq}'}"}(}"N$~~y}#A!}!r} }4}"}&} }`.

23.1.3 Running `pppd`

If you run the `pppd` command above, you will get output something like this:

```
send (AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0^M)
expect (OK)
AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0^M^M
OK
 -- got it

send (ATDT4068500^M)
expect (CONNECT)
^M
ATDT4068500^M^M
CONNECT
 -- got it

send (^M)
expect (name:)
 45333/ARQ/V90/LAPM/V42BIS^M
Checking authorization, Please wait...^M
username:
 -- got it

send (psheer^M)
expect (assword:)
psheer^M
password:
 -- got it

send (??????)
expect (con:)
^M
^M
c2-ctn-icon:
 -- got it

send (ppp^M)
Serial connection established.
Using interface ppp0
Connect: ppp0 <--> /dev/ttyS0
sent [LCP ConfReq id=0x1 <asyncmap 0x0> <magic 0x88c5a54f> <pcomp> <accomp>]
rcvd [LCP ConfReq id=0x3d <asyncmap 0xa0000> <magic 0x3435476c> <pcomp> <accomp>]
sent [LCP ConfAck id=0x3d <asyncmap 0xa0000> <magic 0x3435476c> <pcomp> <accomp>]
rcvd [LCP ConfAck id=0x1 <asyncmap 0x0> <magic 0x88c5a54f> <pcomp> <accomp>]
sent [IPCP ConfReq id=0x1 <addr 192.168.3.9> <compress VJ 0f 01>]
sent [CCP ConfReq id=0x1 <deflate 15> <deflate(old#) 15> <bsd v1 15>]
rcvd [IPCP ConfReq id=0x45 <addr 168.209.2.67>]
sent [IPCP ConfAck id=0x45 <addr 168.209.2.67>]
rcvd [IPCP ConfRej id=0x1 <compress VJ 0f 01>]
sent [IPCP ConfReq id=0x2 <addr 192.168.3.9>]
rcvd [LCP ProtRej id=0x3e 80 fd 01 01 00 0f 1a 04 78 00 18 04 78 00 15 03 2f]
rcvd [IPCP ConfNak id=0x2 <addr 196.34.157.131>]
sent [IPCP ConfReq id=0x3 <addr 196.34.157.131>]
rcvd [IPCP ConfAck id=0x3 <addr 196.34.157.131>]
local  IP address 196.34.25.95
remote IP address 168.209.2.67
Script /etc/ppp/ip-up started (pid 671)
Script /etc/ppp/ip-up finished (pid 671), status = 0x0
 Terminating on signal 2.
Script /etc/ppp/ip-down started (pid 701)
sent [LCP TermReq id=0x2 "User request"]
rcvd [LCP TermAck id=0x2]
```

You can see the expect–send sequences working, so its easy to correct if you made a mistake somewhere.

At this point you might want to type `route -n` and `ifconfig` in another terminal:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
168.209.2.67    0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         168.209.2.69    0.0.0.0         UG    0      0        0 ppp0
```

```
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:2547933 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2547933 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0

ppp0      Link encap:Point-to-Point Protocol
          inet addr:196.34.25.95  P-t-P:168.209.2.67  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
```

```
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
```

This clearly shows what pppd has done: both creating a network device as well as a route to it.

If your name server is configured, you should now be able to ping metalab.unc.edu or some well known host.

23.2 Dial on demand

Dial-on-demand really just envolves adding the demand option to the pppd command-line above. The other way of doing dial-on-demand is using the diald package, but here we discuss the pppd implementation.

With the demand option, you will notice that spurious dial-outs take place. You need to add some filtering rules to ensure that only the services you are interested in cause a dial-out. This is not ideal since there is still the possibility of other services connecting on ports outside of the 1-1024 range. In addition you should also make sure there are no services running except the ones you are interested in.

A firewall script might look as follows. This uses the old ipfwadm command possibly called /sbin/ipfwadm-wrapper on your machine[1] The ports 21, 22, 25, 53, 80, 113 and 119 represent ftp, ssh (Secure Shell), smtp (Mail), domain (DNS), www, auth and nntp (News) services respectively. The auth service is not needed, but should be kept open so that connecting services get a failure instead of waiting for a timeout. You can comment out the auth line in /etc/inetd.conf for security.

```
# enable ip forwarding and dynamic address changing
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/ipv4/ip_dynaddr

# clear all firewall rules
/sbin/ipfwadm -O -f
/sbin/ipfwadm -I -f

# Allow all local comms
/sbin/ipfwadm -O -a accept -P tcp -D 192.168.0.0/16
/sbin/ipfwadm -O -a accept -P udp -D 192.168.0.0/16
/sbin/ipfwadm -O -a accept -P tcp -D 127.0.0.0/24
/sbin/ipfwadm -O -a accept -P udp -D 127.0.0.0/24
/sbin/ipfwadm -I -a accept -P tcp -D 192.168.0.0/16
/sbin/ipfwadm -I -a accept -P udp -D 192.168.0.0/16
/sbin/ipfwadm -I -a accept -P tcp -D 127.0.0.0/24
/sbin/ipfwadm -I -a accept -P udp -D 127.0.0.0/24

# allow ports outgoing
/sbin/ipfwadm -O -a accept -P tcp -D 0.0.0.0/0 20 21 22 25 53 80 119
/sbin/ipfwadm -O -a accept -P udp -D 0.0.0.0/0 53

# restrict all other ports outgoing
/sbin/ipfwadm -O -a deny -P tcp -D 0.0.0.0/0 1:1024
/sbin/ipfwadm -O -a deny -P udp -D 0.0.0.0/0 1:1024

# allow ports incoming
/sbin/ipfwadm -I -a accept -P tcp -D 0.0.0.0/0 22 113

# restrict all other ports
/sbin/ipfwadm -I -a deny -P tcp -D 0.0.0.0/0 1:1023
/sbin/ipfwadm -I -a deny -P udp -D 0.0.0.0/0 1:1023

# deny anything else
/sbin/ipfwadm -I -a deny -P icmp -D 0.0.0.0/0
/sbin/ipfwadm -O -a deny -P icmp -D 0.0.0.0/0
```

IP masquerading can be done with:

```
# Masquerade the domain 192.168.2.0/255.255.128.0
/sbin/ipfwadm -F -f
```

[1] The newer ipchains command is soon to be superceded by a completed different packet filtering system in kernel 2.4 hence I see no reason to change from ipfwadm at this point.

```
/sbin/ipfwadm -F -p deny
/sbin/ipfwadm -F -a m -S 192.168.0.0/17 -D 0.0.0.0/0
```

The pppd script becomes (note that you need pppd-2.3.11 or later for this to work as I have it here):

```
pppd connect \
             "chat -S -s -v \
             '' 'AT S7=45 S0=0 L1 V1 X4 &c1 E1 Q0' \
             OK ATDT<tel-number> CONNECT '' \
             name: <username> assword: '\q<password>' \
             con: ppp" \
    /dev/ttyS0 57600 debug crtscts modem lock nodetach \
    hide-password defaultroute \
    user <username> \
    demand \
    :10.112.112.112 \
    idle 180 \
    holdoff 30
```

23.3 Dynamic DNS

(See also Chapter 22 for other named setups, and Chapter 21 for configuring your machine's DNS lookups.)

Having pppd give IP connectivity on demand is not enough. You also need to your DNS configuration to change dynamically to reflect the current IP address that your ISP would have assigned you.

Now on creation of a connection, pppd runs /etc/ppp/ip-up, which in turn runs /etc/ppp/ip-up.local. Creating /etc/ppp/ip-up.local as the following script, correctly sets up bind. This script assumes that you have one eth0 interface with the IP 192.168.1.1 and that this interface is the gateway for a LAN of four machines masq-a, masq-b, masq-c and masq-d. The domain name of your LAN should be some non-existing name like my-lan.priv.

```
#!/bin/sh

#       $1            $2        $3       $4                $5           $6
# interface-name tty-device speed local-IP-address remote-IP-address ipparam

mkdir /etc/named-dynamic/ >& /dev/null

SERIAL=`expr 2000000000 + \`date '+%s'\` / 10`
IP=$4
ARPA=`echo $IP | cut -f4 -d.`.`echo $IP | cut -f3 -d.`.`echo $IP | cut -f2 -d.`.`echo $IP | cut -f1 -d.`
NAMESERVER=<name-server-of-your-isp>
HOST=`hostname | cut -f1 -d.`
DOMAIN=`hostname | cut -f2,3,4,5,6 -d.`

cat > /etc/resolv.conf <<EOF
search $DOMAIN
nameserver 127.0.0.1
options timeout:18 attempts:4
EOF

cat > /etc/host.conf <<EOF
order hosts,bind
multi on
EOF

cat > /etc/named.conf <<EOF
options {
    forwarders { $NAMESERVER; }; directory "/etc/named-dynamic/";
    dialup yes; notify no; forward only
};
zone "."                        { type hint;    file "named.ca";         };
zone "0.0.127.in-addr.arpa"     { type master;  file "named.local";      };
zone "1.168.192.in-addr.arpa"   { type master;  file "named.192.168.1"; };
zone "$ARPA.in-addr.arpa"       { type master;  file "named.$IP";        };
zone "$DOMAIN"                  { type master;  file "named.$DOMAIN";    };
EOF

cat > /etc/named-dynamic/named.local <<EOF
@       IN      SOA     localhost. root.localhost.  ( $SERIAL 28800 14400 3600000 345600 )
        IN      NS      localhost.
1       IN      PTR     localhost.
EOF

cat > /etc/named-dynamic/named.ca <<EOF
.                       3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000      A     198.41.0.4
.                       3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.     3600000      A     128.9.0.107
.                       3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.     3600000      A     192.33.4.12
.                       3600000      NS    D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.     3600000      A     128.8.10.90
.                       3600000      NS    E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.     3600000      A     192.203.230.10
.                       3600000      NS    F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.     3600000      A     192.5.5.241
.                       3600000      NS    G.ROOT-SERVERS.NET.
G.ROOT-SERVERS.NET.     3600000      A     192.112.36.4
.                       3600000      NS    H.ROOT-SERVERS.NET.
```

```
H.ROOT-SERVERS.NET.      3600000      A     128.63.2.53
.                        3600000      NS    I.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.      3600000      A     192.36.148.17
.                        3600000      NS    J.ROOT-SERVERS.NET.
J.ROOT-SERVERS.NET.      3600000      A     198.41.0.10
.                        3600000      NS    K.ROOT-SERVERS.NET.
K.ROOT-SERVERS.NET.      3600000      A     193.0.14.129
.                        3600000      NS    L.ROOT-SERVERS.NET.
L.ROOT-SERVERS.NET.      3600000      A     198.32.64.12
.                        3600000      NS    M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.      3600000      A     202.12.27.33
EOF

cat > /etc/named-dynamic/named.$IP <<EOF
@       IN      SOA     localhost. root.localhost.  ( $SERIAL 28800 14400 3600000 345600 )
        IN      NS      $HOST.$DOMAIN.
        IN      PTR     $HOST.$DOMAIN.
EOF

cat > /etc/named-dynamic/named.192.168.1 <<EOF
@       IN      SOA     localhost. root.localhost.  ( $SERIAL 28800 14400 3600000 345600 )
        IN      NS      $HOST.$DOMAIN.
1       IN      PTR     $HOST.$DOMAIN.
2       IN      PTR     masq-a.$DOMAIN.
3       IN      PTR     masq-b.$DOMAIN.
4       IN      PTR     masq-c.$DOMAIN.
5       IN      PTR     masq-d.$DOMAIN.
EOF

cat > /etc/named-dynamic/named.$DOMAIN <<EOF
@       IN      SOA     localhost. root.localhost.  ( $SERIAL 28800 14400 3600000 345600 )
        IN      NS      $HOST.$DOMAIN.
$HOST   IN      A       $IP
masq-a  IN      A       192.168.1.2
masq-b  IN      A       192.168.1.3
masq-c  IN      A       192.168.1.4
masq-d  IN      A       192.168.1.5
EOF

killall -1 named
```

The options `dialup yes; notify no; forward first` tell `bind` to use the link as little as possible; not send notify messages (there are no slave servers on our LAN to notify; and try forward requests to the name server under `forwarders` before trying to answer them itself; respectively.

There is one problem with this configuration. Queued DNS requests are flushed when the configuration is reread with `killall -1 named`. When you try, say `ftp sunsite.unc.edu`, the first DNS request by `ftp` causes a dial-out, but then is discarded. The next DNS request (18 seconds later — `options timeout:18 attempts:4`) also doesn't make it (dial-outs take more than 30 seconds on my machine). Only the third request gets through. What is really needed is a DNS program designed especially for masquerading dynamically-assigned-IP servers.

The above scripts are probably over-kill, so use them sparingly. For example, there is probably no application that really needs forward and reverse lookups on the `ppp0` device, hence you can do with a DNS configuration that doesn't need restarting on every dial-out. The `bind` documentation promises better support for dialup servers in the future.

There is a further option: that is to use the `dnrd`, a DNS package especially for dial-out servers. It was not created with dial-on-demand in mind though, hence it has some limitations.

23.4 Using `tcpdump` to watch your connection

If a dial out does occur unexpectedly, you can run `tcpdump` to dump packets going to your `ppp0` device. This will probably highlight the error. You can then look at the TCP port of the service and try to figure out what process the packet might have come from. The command is:

```
tcpdump -n -N -f -i ppp0
```

`tcpdump` is also discussed in Section **??**.

23.5 Using ISDN instead of Modems

A lot of companies will see a regular modem as the best way to get connected to the Internet. Because ISDN is considered esoteric, they may have not looked at it as an option. In fact ISDN is preferable everywhere except for single user dial-up (i.e. home use).

For those who are not familiar with ISDN, this paragraph will give you a quick summary. ISDN stands for *Integrated Services Digital Network*. ISDN lines are like regular telephone lines, except that an ISDN line comes with two analogue and two digital channels. The analogue channels are regular telephone lines in every respect

— just plug your phone in and start making calls. The digital lines each support 64 kilobits/second data transfer — only ISDN communication equipment is meant to plug into these. To communicate over the digital line you need to dial an ISP just like with a regular telephone. Now it used to be that only very expensive ISDN routers could work with ISDN, but ISDN modems and ISDN ISA/PCI cards have become cheap enough to allow anyone to use ISDN, while most telephone companies will install an ISDN line as readily as a regular telephone line. So you may ask whats with the "Integrated Services". I suppose it was thought that this service, in both allowing data as well as regular telephone, would be the ubiquitous communications service. This remains to be seen.

If you have a hundred ISDN boxes to setup, it would be well worth it to buy internal ISDN cards: they are really low priced these days. Configuring these is not covered here for now. However, if you have one ISDN box to configure and no clue about ISDN, an internal card is going to waist your time. In this case a ISDN external modem is the best option. These are devices designed as drop in replacements to a normal external modem — they plug into your serial port and accept (probably ignore) the same AT command strings as a normal modem.

Although these are *supposed* to be drop in replacements, ISDN *is* a completely different technology. In particular, there are different protocols for different countries which have to be specified to the card. I myself have not ever had to set up ISDN, but the advice I get so far is:

For an Asyscom modem running on a particular ISP here in South Africa we had to enter the AT commands:

```
ATB4
ATP=17
```

and also add `asyncmap 0x00000000` to the `pppd` command-line.

Also `ATDTXXXXXXX` should become just `ATDXXXXXXX`.

Another source of info recommends for Zyxel modems `ATB20` for V.120 Standard LAPD and euro DSS1 protocols, and `ATB40` for 64K sync PPP used with CISCO ISP equipment. And then also

```
AT&ZI4=0
AT&W0
ATZ0
```

This should give you an idea of what you may have to change to get ISDN working, it is by no means a product endorsement.

Be weary when setting up ISDN. ISDN dials **really** fast. It can dial out a thousand times in a few minutes which is expensive.

Chapter 24

Case Example A with General Recommendations in the Field

Although a theoretical understanding of UNIX is entertaining reading, most readers will be interested in how to apply their configuration knowledge in an actual situation. What is typical is that a would-be administrator has *no-idea-where-to-start*.

This seems to be a bottleneck in the learning of UNIXadministration — few are willing to coach the student in real life situations.

The case studies given here will give an idea of how this all fits together in practice.

24.1 University of the Western Cape, Laboratories, South Africa

The University of the Western Cape (UWC) pharmacy department conduct experiments on frogs legs. They stuff some chemical into the frog (or other) muscle tissue and measure the contraction. Their in-house electronics shop has produced force meters (*transducers*) and have an Eagle Electronics PC30F Universal IO ISA card (otherwise known as a multi-channel Analogue to Digital Converter ADC).

In other words, the force meter sends a signal to this ISA card which can then be read digitally by some software.

They have no computer facilities at all in this laboratory. They require some setup that each of eight students can monitor their own experiment simultaneously. They also need their post-graduate research programs to be able to monitor 16 simultaneous experiments from one machine.[1]

I suggested a network of machines dual booting LINUX and Windows (they require to run other software). Their original idea was to buy an ADC card for *each* machine. These are very expensive. Of course it would be rather trivial to read the data over a network. The contractions themselves need to be sampled at 1 second intervals only.

The project looked like it had four distinct parts:

Networking and installation of PC's.

Installation and configuration of LINUX.

Writing of the low level ADC driver.

Writing of a graphical interface to read ADC values over the network and display them.

[1] UWC originally approached the biomedical research department of the *University of the Cape Town*, requesting a student to write software to monitor their experiments. Of course, as is always the case with clients, they did not anticipate the breadth of the project — it is not a *software* project per se. A friend of mine got wind of the project, which then came to me informally.

24.2 Spec and quote

Most companies require detailed specifications and contracts to be drawn before entering a project. This however can consume as much time as the project itself[2]. I prefer to loosely specify what the system has to do, then gauge intuitively the amount of time it will take to meet the *expectation* of the client, regardless of what the client *thinks* the system needs to do.

This means having to estimate what the client wants to *feel* about the completed project. You must tactfully discuss their vision to read the result you yourself would like in their position.

It is also important to define to yourself what the client *doesn't know*, without making them feel insecure about their ignorance. Also remember that its no use trying to teach your client, although any information you give them must always open up new possibilities, and never close them.

Clients should never be told what isn't possible. Present options rather than limitations, but never confuse them with decisions.

> *You must give yourself enough leeway to implement the little features that the client never thought of, and enough extra time to make things generic enough to be reused for other clients. Never do the absolute minimum.*

In this case, the responsible person merely had an idea that his current processes needed computer automation. To negotiate a specification would have been too tedious. He imagined an application that would calibrate, graph and save recorded data. Because I know that it is simple enough to come up with a generic application that does more than he needs, I am not going to go into details about what it doesn't need to do.

The project was quoted at 60 hours. Our company rates are rather cheep — $40 per hour (at Jan 2000) excluding VAT (i.e. tax).

24.3 Dealing with Universities

Dealing with Universities (Colleges) is different from dealing with small companies. Some points to note are:

Universities are very large organisations. They are necessarily very bureaucratic. Although this may seem like a hindrance to your job, you must realise that every task conceivable has a person assigned to it, and a person to supervise that person. To get anything done, its just a matter of finding out which person *is* responsible. Any line of questioning should aim to try find out the conventions of delegation, rather than actually trying to get anything done.

Any organisation of any appreciable size will have its own IT department.

An IT department's main activity is telling people what to click on when things don't work. Hence most of the staff are trained in this area only. Under no circumstances question them on an issue they don't understand it will create unnecessary confusion.

An IT department will *never* have more than one person who actually knows how the whole LAN fits together. Because the remainder of the department doesn't need to, they won't know who that person is either.

Any appreciably sized organisation will have a single supplier of computer hardware.

Universities don't need to show a profit. The people they employ rarely have business experience. These people are afraid on one hand, and careless on the other.

The IT department may only know about proprietary software.

[2] This is obviously not true with very large projects, but then you would charge for the drawing up and negotiation of the spec as well.

The question to ask to find out any issue about the current network setup is simple: "Who is in charge of your DNS server". This person will necessarily know enough about the LAN to at least tell you who else to speak to. Don't be afraid to go hunting for that person on foot — he may be on lunch.

Another question to ask is "I need a new network point — who do I speek to?". Asking questions that **have been asked before** is the key, even if that's not what you really need done. In this case I needed to know about the routing — a person who would install network points would necessarily know who would know about routing.

Finally, you should be careful to always show that you have the University's best interests at heart. Businessman have an ethic of balancing risk, price and good-will. Academia, however, are naive and need more to feel secure.

24.4 Networking 8 Machines

The hardware was supplied by the IT department, which did a very nice job of installing an excess of network points, and running Windows in a first partition. The network uses DHCP and has a DHCP server.

Because the LINUX boxen need to communicate with each other, they need to know their IP addresses. DHCP addresses expire if the machine has been inactive for some time. Hence the LAN needed to have statically allocated IP addresses. This was a good excuse to put the LAN on its own segment (i.e. having a dedicated hub, with one machine having two network cards to route to the rest of the network.)

In general, it is a very bad idea to try get LINUX boxes working on the same segment as many Windows machines. Try to get your LINUX LAN on its own segment at all costs — in fact, it is better to make it a requirement of the project. In the first place, it enables you to firewall the segment for security, and second it enables you to deal exclusively with TCP traffic. (This does mean IPX will not be able to communicate with the rest of the LAN by default.) Finally, from an administration point of view, you can setup your Linux boxes anyway you like, and have control over mail and DNS, since only one machine has to interface with the rest of the LAN. You then never have to deal with someone else's DNS/Mail/etc. server being down. When you have network problems, you can be sure they are *yours*.

The IT department may also not know that networks can be isolated from each other with routers made from old 486 boxes, or in this case, using one of the workstations. Be careful to explain that isolating the segment will cost them nothing extra besides a hub and a network card.

24.5 Installation and configuration of Linux

The machines each had stock standard hardware (fortunately). Hardware *can* be a major problem. Although a large organisation will probably be bent on particular brands (usually the popular well supported ones), you may run into a flock of machines that have an unsupported video card. Don't try explain to them that they need to be sure that the hardware is supported beforehand — usually the supplier won't even know. Its better to get them to order it anyway, with the warning that extra hardware may need to be purchased. Most suppliers have a 7 or 30 day money back guarantee, so you can get those graphics cards replaced with (possibly slightly more expensive) other ones.

Some things *to* specify are:

Graphics cards must be exactly 2 Megabytes. (Over 2 megabytes is silly — see Section ***). This is because you want to run 1024x768 in 16 bit colour. A true colour display at 1024x768 is essential for X.

Mice *must* be three button for X.

Machines must be entry level (bottom of the range). Since any even entry level machine is a huge excess for running Linux, there is no point to anything else.

Order no less that 32 megabytes RAM for decent X performance.

> *Never underestimate the time it is going to take you to configure hardware and basic OS setup. Its the little basic things that can take days when you have lots of machines.*

The machines were set up as follows:

Machines were numbered 1 through 8 and given IP address `192.168.1.`*machine*.

Machine 1 was designated a "server". It was setup so that reverse and forward lookups worked for the `192.168.1.`*???* IP addresses. Reverse and forward lookups on the local LAN is an essential first step.

`rsh` was set up on each machine so that commands could be issued to the whole LAN to make further administration easy. A script was written to `rsh` a command to all machines.

YP services where enabled (NIS: see the `NIS-HOWTO`) so that logins could be made anywhere on the LAN.

NFS was setup so that the `/home` directory would be mounted on boot from the server.

X was setup as the default run level.

IP forwarding and masquerading was enabled on the server. Its second network card was given a static IP address issued from the IT department. This enabled all machines to access the web.

With the server having an InkJet printer attached, *LPRng* `<http://www.astart.com/lprng/LPRng.html>` was installed for printing. I had difficulty getting ordinary `lpd` to work, so this was installed instead. This package also allows the print filter to reside on the server only.

The LAN was now setup for use. Users could be added on the server, and seamlessly login anywhere, and use web, and office applications, with print jobs defaulting to the right place.

24.6 Writing of the low level ADC driver

The ADC card came with plenty of detailed documentation. To write a driver for new hardware requires intimate knowledge of C and a good idea of kernel programming. However, I had telephoned the manufacturers (who were conveniently in the same city) who had explained that a polling driver was relatively simple.

The excellent documentation provided with the card allowed me in a few hours to write C code to read ADC values from the test apparatus provided by UWC's Pharmacy workshop.

The next step was to write a daemon program to serve these values to the network. The listing is given in Appendix *** (390 lines). The program uses a single thread in order not to have two connections doing IO operations on the card simultaneously. The values it serves are textual in order that it can be tested by easily `telnet`ing to the port and typing in channel requests. The code effectively allows any channel to be read from anywhere, hence the same channel can really be read by many remote processes.

24.7 Writing of a graphical interface

The `gnome-python` package that comes with *Gnome* provides an example use of *Gnome Canvas*. Gnome Canvas is a Gnome widget that allows arbitrary figure objects to be drawn and manipulated in a window. It is ideal for creating a simple graphing package. Python socket calls can then read ADC values by connecting to the server.

The listing for this program is given in Appendix *** (480 lines).

The program can graph multiple channels simultaneously. Hence UWC would be able to meet both their requirements of monitoring multiple channels on one machine or single channels on several machines. It automatically scales the graphs to fit in the window and allows the sample frequency to be set. Graphs can be saved as tab separated lists for import into a spread sheet.

The program also does least squares fitting on a line of calibration points allowing the transducer to be accurately calibrated.

24.8 Specifications

The nice thing about free software is that I can sell me little installation over and over now that I have done it. Here is the complete specification that I might advertise to other clients:

Obsidian Network IO Utility^{TM}

Product Summary: Allows a single multichannel ADC card to support simultaneous data capturing programs running remotely on a TCP/IP network. A single ADC card can support multiple remote capture points to allow data to be monitored across a network.

Hardware requirements: Eagle PC30F. Entry level PC's, or at least PIII 300 MHz. Non-PC Unix workstations on special request.

Software requirements: All software included. No requirements.

Application functions: Real time graphing of multiple channels. Calibration. Auto scaling of data to window size. Up to 16 channels.

Data rate: 400Hz for single channel view; 100Hz for 4 channel view; 50Hz for 8 channels view. Higher rates on special request up to the limits of the IO card and network.

Data save format: Time stamped tab delimited ASCII. Other formats on special request.

Server: Supports an arbitrary number of client applications, each reading multiple channels over TCP/IP. Multiple clients can read the same channels.

Development platform: GNU/Linux with Gnome GUI.

Chapter 25

Case Example B

An certain IT company services the needs of a large number of retail vendors.

They have been running SCO Unix for some time, but switched to LINUXwhen they discovered its seemless SCO binary emulation.

Their current situation is several hundred dial-in machines spread throughout South Africa and neighbouring countries. Each one runs a LINUXmachine with several WYSE[1] character terminals and one or more character printers. They run a proprietary package called Progress which they have licensed at enormous cost. Progress is a database system with a character interface that serves as a Point-Of-Sale terminal and accounting solution.

Each machine has a rather arcane menu interface that does various common administration functions.

The machines are scripted to nightly communicate with a centralised server. Somewhere a further centralised IBM AS/400 also comes into play.

The company desires to have an X Windows interface on all their machines. With this, they want to enable mail and web access for all the vendors.

The existing text based interface to Progress must also be preserved through an X terminal emulator.

They themselves have a call centre and intend to install, ship and support the machines. I was to provide an master installation machine from where many machines could be replicated.

25.1 Overall Configuration

The machines run on or more *Chase* ISA serial port cards[2]. These required configuration on boot. The only other specialised hardware was the "LS120" floppy disks that the company uses for backup purposes (instead of tapes). These are ATAPI IDE floppy disks that support 1.44 Megabyte 3.5' floppy disks as well as 120 Megabyte floppy disks — basically involving symlinking `/dev/fd0` to `/dev/hdc`.

X Windows was configured to run in 1024x768 at 16 bit colour.

The machines were enabled for dialup to an ISP. Their ISP provided a single nationwide telephone number. The machines are intended for a single user to sit at the console and use the Internet, so I recommended against dial-on-demand — rather give the user power to dial in or hang up as necessary, in full knowledge that they are accruing a telephone bill.

The text based menu system system was rewritten with `dialog` and `gdialog`. Hence users logging in via terminals would see the identical interface (albeit text-based) to those logging in on the X console. The text menu system would execute as the last script in `/etc/profile.d` provided they were attached to a terminal (i.e. not an X login) (see `man tty`). The `icewm` window manager has a clean `menu` file that allows configuration of the window managers "Start" menu. Selections under the "Start" button are identical to the text menu seen when logging in via a character terminal. A nice trick is to write a C program front end to `gdialog` and `dialog` that uses the `ttyname(0)` system call to check if it is attached to a terminal, and then invokes the correct program. I called this new command `vdialog`.

[1] Brand of character terminal.
[2] The preferred brand of the client.

Users also have several different privilege levels, and each should only see their allowed menu selections. Hence the menus are created dynamically on login. Note that the users' ignorance of the system was considered an adequate security measure — their are no *real* privilege levels.

A Netscape `/.netscape` was created to be copied into the users home directory with his personal settings ready on first login.

Dial-ins were accompanied by fetching a user list from a master server; adding the appropriate users; then fetching their mail by pop using `fetchmail`; and simultaneously doing `exim -qf` to flush the pending mail queue. The master server acts as a mail, pop and DNS server.

`exim` was configured to deliver local mail immediately and defer outgoing mail. In addition message size was limited to 2 megabytes and an outgoing MIME filter was written to strip audio, video and image attachments. The same filter is applied on the server side for incoming messages.

`printcap` was configured with `lp` as the default printer for their dot-matrix printers, using a filter created by `printtool`. A second printer, `fax`, sent jobs through a custom filter that brought up a `gdialog --inputbox` on the current X display to request a telephone number from the user. It then converted the PostScript file (through several different formats as needed) into the appropriate format for `sendfax`. Unfortunately, at the time of implementation, `sendfax` did not support sending the 'From' telephone number in the fax protocol, but this was acceptable to the client. I also added a `.printer` file to the users home directory. This was read on startup to set the `PRINTER` environment variable. Users had a menu option to set this to any of `/dev/ttyS4` through `/dev/ttyS20`.

Finally, `uucp` was setup for incoming calls, since an additional master server existed that would expect the vendor to answer and provide a uucp login. An nice trick is to run `uuchk` which dumped the config on the previous older system. When the `uuchk` output on the new system matched the old, we could be sure that things were cool with the `uucp` setup.

25.2 `setuid` scripts and security

There are several areas where writing a C program was unavoidable or just too convenient to pass over. The `vdialog` utility mentioned above was one.

Users have a menu selection to *execute-a-dial-in-and-exchange-mail*. This runs a generic script which dials in and runs `fetchmail` and `exim -qf`. The script is one of a few that must obviously run as `root`.

The simplest solution is to make a C program that has its sticky bit set with `chown a+s <filename>`. The program itself is simply:

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main (int argc, char **argv)
{
    int p;
    setuid (geteuid ());         /* Set the actual user and group */
    setgid (getegid ());         /* to that of the sticky ones. */
    p = system ("/etc/ppp/dial.sh >> /var/log/fetchmail 2>&1");
    if (p)
        exit (p);
    p = system ("/etc/fetchmail/exchange_mail.sh");
    exit (p);
    return 0;   /* prevent gcc warning */
}
```

You can see that this merely runs the existing shell scripts `/etc/ppp/dial.sh` and `/etc/fetchmail/exchange_mail.sh` albeit as root.

This is a standard trick that is often necessary to get around complex manipulations with groups. It does however create a security hole. However, in the present circumstance, all dial in machines are `192.168.*.*` addresses, which are firewalled in from the Internet, hence security subtleties are not a huge issue. As a second

point, as a security hole, it is small. All the attacker can really do is execute dial-in's. On the other hand, **never** create such a "setuid" program that can execute an arbitrary script. An example of such a program is:

```
int main (int argc, char **argv)
{
    setuid (geteuid ());
    setgid (getegid ());
    system (argv[1]);
}
```

which is a **gaping** security hole. In other words, if all setuid programs have a single immutable function, and also take no options or input, then you are safe.

25.3 Custom rxvt for Progress

The Progress package runs perfectly on one machine inside a terminal emulator, but on other newer machines, it fails with an ...cannot redirect output... error message, although it runs perfectly on the console on all machines.

Of course it is typical of proprietary software to have such a limitation. What it is doing is not immediately obvious.

It requires some brilliance to deduce that Progress had decided that it was not on a terminal based on the return value of the ttyname() system call. It was not merely checking if the system call failed with a NULL return value, but *also* checking if the first part of the terminal device matched the text /dev/tty. At some point RedHat/rxvt switched to the /dev/pts/ system of pseudo terminal devices, hence the failure.

To solve the problem, I got the source for the rxvt terminal emulator. Within it, it uses several different methods of getting a pseudo terminal, depending on the system it was compiled for. It also supports the older BSD style pseudo terminals, /dev/tty, that Progress required. By simply adjusting the config.h file to

```
/* Define possible pty types */
/* #undef PTYS_ARE_NUMERIC */
/* #undef PTYS_ARE_PTMX */
/* #undef PTYS_ARE_PTC */
/* #undef PTYS_ARE__GETPTY */
/* #undef PTYS_ARE_GETPTY */
/* #define PTYS_ARE_GETPT 1 */
/* #undef PTYS_ARE_CLONE */
#define PTYS_ARE_SEARCHED 1
```

and recompiling, Progress ran as normal.

This is obviously an obscure and not widely applicable technique, but serves to show the sort of things you might have to resort to as a result of poor proprietary development.

25.4 Mail server

exim ran on the mail server as well. This machine hosts a per machine user list as /<hostname>/user. I chose the userlist to have the format <login-name>:<plain-text-password>:<full-name> similar to a passwd file.

It is from these user lists that the client dial-in machines create their own user logins. Simple use of cut and sed and a new utility setpasswd will create the proper account on the local machine.

Chapter 26

Corporate Frequently Asked Questions

This FAQ was compiled in August 1999. The rapid pace of IT[1] *means that it may well be out of date by the time you read it. Please consult the various Internet resources listed for more up to date information.*

This FAQ is not essential reading for Rute and you can skip questions you do not find interesting

26.1 Linux Overview

This section covers questions that pertain to Linux as a whole.

26.1.1 What is Linux?

Linux is the core of a free Unix operating system for the PC and other hardware platforms. Developement of this operating system started in 1984; called the GNU project of the Free Software Foundation (FSF). The Linux core (or kernel) is named after its author, Linus Torvalds. It began development in 1991 - the first usable releases where made in 1993. Linux is often called GNU/Linux because much of the OS is comprised of the efforts of the GNU project.

Unix systems have been around since the 1960's and are a proven standard in industry. Linux is said to be POSIX compliant, meaning that it confirms to a certain definite computing standard laid down by academia and industry. This means that Linux is largely compatible with other Unix systems (the same program can be easily ported to run on another Unix system with few (sometimes no) modifications) and will network seamlessly with other Unix systems.

Some commercial Unix systems are IRIX (for the Silicon Graphics); Solaris or SunOS for Sun Microsystem's SPARC workstations; HP Unix for Hewlett Packard's servers; SCO for the PC; OSF for the DEC Alpha machine and AIX for the PowerPC/RS6000.

Some freely available Unix systems are NetBSD, FreeBSD and OpenBSD and also enjoy widespread popularity.

Unix systems are multitasking and multiuser systems - meaning that multiple concurrent users running multiple concurrent programs can connect to and use the same machine.

26.1.2 What are Unix systems used for? What can Linux do?

Unix systems are the backbone of the Internet. Heavy industry, mission critical applications, and universities have always used Unix systems. High end servers and multiuser mainframes are traditionally Unix based. Today Unix systems are used by large ISP's through to small businesses as a matter of course. A Unix system is the standard choice when a hardware vendor comes out with a new computer platform because Unix is most amenable to being ported. Unix systems are used as database, file, and Internet servers. Unix is used for visualization and graphics rendering (like some Hollywood productions). Industry and universities use Unix systems for scientific simulations.

[1] Information Technology

The wide spread use of Unix is not well advertised because of a failure on the part of the media, and because Unix systems are unjustifiably thought to be more expensive and complicated, and therefore not suited for mainstream audiences. This does not offset the fact that most mission critical servers in existence are Unix systems.

Linux can operate as a web, file, smb (WinNT), Novell, printer, ftp, mail, sql, masquerading, firewall, and pop server to name but a few. It can do anything that any other network server can do faster and more reliably.

Linux's up and coming graphical user interfaces are of the most functional and aesthetically pleasing ever to have graced the computer screen. Linux has now moved into the world of the desktop.

26.1.3 What other platforms does it run on including the PC?

Linux runs on

- 386/486/Pentium Processors.
- Digital Alpha's 64 bit processors.
- Motorola's 680x0 processors, included Commodore Amiga, Atari-ST/TT/Falcon and HP Apollo 68K
- Sun Sparc workstations, including sun4c and sun4m as well as well as Sun4d and Sun4u. Multiprocessors machines are supported as well as full 64 bit support on the Ultrasparc.
- Advanced Risc Machine (ARM) processors.
- MIPS R3000/R4000 processors including Silicon Graphics machines.
- PowerPC machines.
- Merced suport is promised.

Other projects are in various stages of completion - eg, you may get Linux up and running on many other hardware platforms, but it would take some time and expertise to install, and you may not have graphics capabilities. Every month or so one sees support announced for some new esoteric hardware platform. Watch the Linux Weekly News *lwn.net* `<http://lwn.net/>` to catch these.

26.1.4 What is meant by GNU/Linux as opposed to Linux?

(See also 'What is GNU?' below and 'What is Linux?' above)

In 1984 the Free Software Foundation (FSF) set out to create a free Unix-like system. It is only because of their efforts that the many critical packages that go into a Unix distribution are available. It is also because of them that a freely available, comprehensive, legally definative, free-software license is available. Because many of the critical components of a typical Linux distribution are really just GNU tools developed long before Linux, it is unfair to call any distribution a 'Linux' system. The term GNU/Linux is more accurate and gives credit to the larger part of Linux.

26.1.5 What web pages should I look at?

There are hundreds of web pages devoted to Linux. There are thousands of web pages devoted to different free software packages. A net search will reveal the enormous amount of info available.

- Three places for general Linux information are:
  - *www.linux.org.uk* `<http://www.linux.org.uk/>`
  - *www.linux.org* `<http://www.linux.org/>`
  - *Linux International* `<http://www.li.org/>`
- For kernel information see

- *Linux Headquarters* <http://www.linuxhq.com/>

- A very important site is
  - *FSF Home Pages* <http://www.gnu.org/>

  which is the home page of the free software foundation and explains their purpose and the philosophy of software that can be freely modified and redistributed.

- Three large indexes of reviewed free and proprietary Linux software are:
  - *Linuxberg* <http://www.linuxberg.org/>
  - *Linux Mall* <http://www.LinuxMall.com/>
  - *Scientific Applications for Linux (SAL)* <http://SAL.KachinaTech.COM/index.shtml>

- Announcements for new software are mostly made at
  - *www.freshmeat.net* <http://www.freshmeat.net/>

- The Linux weekly news brings up to date info covering a wide range of Linux issues:
  - *lwn.net* <http://lwn.net/>

- The two major Linux desktop projects are:
  - *Gnome Desktop* <http://www.gnome.org/>
  - *KDE Desktop* <http://www.kde.org/>

But don't stop there - there are hundreds more.

26.1.6 What are Debian, RedHat, Caldera and Suse etc. Explain the different Linux distributions?

Linux is really just the 'kernel' of the operating system. Linux in itself is just a 1 megabyte file that runs the rest of the system. Its function is to interface with hardware, multitask and run real programs which do tangible things. All applications, network server programs, and utilities that go into a full Linux machine are really just free software programs recompiled to run on Linux - most existed even before Linux. They are not part of Linux and can (and do) actually work on any other of the Unix systems mentioned above.

Hence many efforts have been taken to package all of the utilities needed for a Unix system into a single collection, usually on a single easily installable CD.

Each of these efforts combines hundreds of 'packages' (eg the Apache web server is one package, the Netscape web browser is another) into a Linux 'distribution'.

Some of the known Linux distributions are:

- *Apokalypse* <http://www.gate.net/~mclinux/>
- *Armed Linux* <http://www.armed.net/>
- *Bad Penguin Linux* <http://web.tiscalinet.it/badpenguin/>
- *Bastille Linux* <http://www.bastille-linux.org/>
- *Best Linux (Finnish/Swedish)* <http://www.bestlinux.net/>
- *Bifrost* <http://www.data.slu.se/bifrost/dist.en.html>
- *Black Cat Linux (Ukrainian/Russian)* <http://www.blackcatlinux.com/>
- *Caldera OpenLinux* <http://www.calderasystems.com>
- *CCLinux* <http://www.CosmicChaos.com/CClinux/>

- *Chinese Linux Extension* <http://cle.linux.org.tw/CLE/e_index.shtml>
- *Complete Linux* <http://www.macmillansoftware.com/catalog/software_bud.cfm?isbn=1575953099>
- *Conectiva Linux (Brazilian)* <http://www.conectiva.com.br/cl/index.html>
- *Debian GNU/Linux* <http://www.debian.org>
- *Definite Linux* <http://definite.ukpost.com/definite.html>
- *DemoLinux* <http://www.demolinux.org/>
- *DLD* <http://www.delix.de>
- *DLite* <http://opensrc.org/dlite/>
- *DLX* <http://www.wu-wien.ac.at/usr/h93/h9301726/dlx.html>
- *DragonLinux* <http://www.dragonlinux.nu/>
- *easyLinux* <http://www.easyLinux.com>
- *Enoch* <http://enoch.masslinux.com/>
- *Eridani Star System* <http://www.amush.cx/linux/>
- *Eonova Linux* <http://www.eonova.com/>
- *e-smith server and gateway* <http://www.e-smith.net>
- *Eurielec Linux (Spanish)* <http://www.etsit.upm.es/~eurielec/linux>
- *eXecutive Linux* <http://www.exelinux.com/>
- *floppyfw* <http://www.zelow.no/floppyfw/>
- *Floppix* <http://www.algonquinc.on.ca/~macewal/floppix/>
- *Green Frog Linux* <http://members.linuxstart.com/~austin/index.html>
- *hal91* <http://home.sol.no/~okolaas/hal91.html>
- *Hard Hat Linux* <http://www.mvista.com/hardhat/>
- *Independence* <http://independence.seul.org/>
- *Jurix* <http://www.jurix.org>
- *Kha0s Linux* <http://www.kha0s.org/>
- *KRUD* <http://www.tummy.com/krud/>
- *KSI-Linux* <http://www.ksi-linux.com>
- *Laetos* <http://www.laetos.org>
- *LEM* <http://linux-embedded.com/>
- *Linux Cyrillic Edition* <http://www.usoft.spb.ru/Graphic/English/Products/products.html>
- *LinuxGT* <http://www.greysite.com>
- *Linux-Kheops (French)* <http://www.linux-kheops.com/kh97-33/index.htm>
- *Linux MLD (Japanese)* <http://www.mlb.co.jp>
- *LinuxOne OS* <http://www.linuxone.net/>
- *LinuxPPC* <http://www.linuxppc.com>
- *LinuxPPP (Mexican)* <http://www.os.com.mx/>

- *Linux Pro Plus* <http://www.LinuxMall.com/Allprod/lxcdr.html>
- *Linux Router Project* <http://www.linuxrouter.org>
- *LOAF* <http://loaf.ecks.org/>
- *LSD* <http://wwwcache.ja.net/dev/lsd/>
- *Mandrake* <http://www.linux-mandrake.com/>
- *Mastodon* <http://www.pell.portland.or.us/~orc/Mastodon/>
- *MicroLinux* <http://linux.hr/microlinux/>
- *MkLinux* <http://www.mklinux.org>
- *muLinux* <http://mulinux.nevalabs.org/>
- *nanoLinux II* <http://www.pluto.linux.it/ildp/AppuntiLinux/AL-13.40.172.html>
- *NoMad Linux* <http://www.nomadlinux.com/>
- *OpenClassroom* <http://www.openclassroom.org/>
- *Peanut Linux* <http://metalab.unc.edu/peanut/>
- *Plamo Linux* <http://www.linet.gr.jp/~kojima/Plamo/>
- *PLD* <http://www.pld.org.pl>
- *Project Ballantain* <http://www.linuxsupportline.com/~router/>
- *PROSA* <http://www.prosa.it/prosa-debian/>
- *QuadLinux* <http://www.quadlinux.com/>
- *Red Hat* <http://www.redhat.com>
- *Rock Linux* <http://www.rock-projects.com/linux_e.html>
- *RunOnCD* <http://www.netian.com/~cgchoi>
- *ShareTheNet* <http://www.ShareTheNet.com>
- *Skygate* <http://www.skygate.co.uk/skylinux.html>
- *Slackware* <http://www.slackware.com>
- *Small Linux* <http://smalllinux.netpedia.net>
- *Stampede* <http://www.stampede.org>
- *Stataboware* <ftp://www.netstat.ne.jp/pub/Linux/Linux-Alpha-JP/>
- *Storm Linux* <http://www.stormix.com>
- *SuSE* <http://www.suse.com>
- *Tomsrtbt* <http://www.toms.net/rb/home.html>
- *Trinux* <http://www.trinux.org/>
- *TurboLinux* <http://www.turbolinux.com>
- *uClinux* <http://ryeham.ee.ryerson.ca/uClinux>
- *Vine Linux* <http://vine.flatout.org>
- *WinLinux 2000* <http://www.winlinux.net/>
- *Xdenu* <http://xdenu.tcm.hut.fi/>
- *XTeamLinux* <http://www.xteamlinux.com.cn>
- *Yellow Dog Linux* <http://www.yellowdoglinux.com/>

There are over a hundred distributions of Linux. Many are just minor variations on an existing distributions.

26.1.7 Who developed Linux?

Linux's leading authors are the Free Software Foundation (FSF) of MIT, and Linus Torvalds.

The Linux kernel CREDITS list has over 200 names in it. These are all people who contributed source code to the kernel. There are also between 200 and 600 packages (See previous question) that go into a typical Linux installation. Most of these have more than one developer - usually a loosely knit team of computer enthusiasts who have never met each other in person - and hence Linux really has thousands of 'authors'.

A *Free Software Who's Who* <http://support.lcg.org/Whoswho/> has been created listing all the big names.

It has been suggested that at over 10000 people are actively developing or testing Linux round the clock.

26.1.8 Why should I not use Linux?

If you are a private individual with no Unix expertise available to help you when you come into problems, AND you are not interested in learning about the underlying workings of a Unix system, then don't install Linux.

If you depend on your desktop programs always looking and working the same way, and you are not interested in the security of your data, or the performance of your machine, then don't install Linux.

If you are not interesting in leaving your computer on week after week and have the desktop remain as you left it day in and day out, without failures or losses in performance, then don't install Linux.

26.2 Linux, GNU and Licensing

This section covers questions about the nature of free software and the concepts of GNU

26.2.1 What is Linux's license?

The Linux kernel is distributed under the GNU General Public License (GPL), available from the Free Software Foundation:

- *FSF Home Page* <http://www.gnu.org/>

Most (95% ?) of all other software in a typical Linux distribution is also under the GPL or the LGPL (see below).

There are many other types of free software licenses. Each of these is based on particular commercial or moral outlooks. Their acronyms are as follows (as defined by the Linux Software Map database) in no particular order:

- PD - Placed in public domain
- shareware - Copyrighted, no restrictions, contributions solicited
- MIT - MIT X Consortium license (like BSD's but with no advertising requirement)
- BSD - Berkeley Regents copyright (used on BSD code)
- Artistic License - Same terms as Perl Artistic License
- FRS - Copyrighted, freely redistributable, may have some restrictions on redistribution of modified sources
- GPL - GNU General Public License
- GPL 2.0 - GNU General Public License, version 2.0
- GPL+LGPL - GNU GPL and Library GPL
- restricted - Less free than any of the above

More info on these licenses can be had from

- ftp://metalab.unc.edu/pub/Linux/LICENSE

26.2.2 What is GNU?

GNU is an acronym for Gnu's Not Unix. A gnu is a large beast and is the motif of the Free Software Foundation (FSF). GNU is a 'recursive' acronym.

Richard Stallman is the founder of the FSF and the creator of the GNU General Public License. One of the purposes of the FSF is to promote and develop free alternatives to proprietary software. The GNU project is an effort to create a free Unix-like operating system from scratch and was started in 1984.

GNU represents this free software licensed under the GNU General Public License. GNU software is software designed to meet a higher set of standards than its proprietary counterparts.

GNU has also become a movement in the computing world. When the word GNU is mentioned, it usually evokes feelings of extreme left wing genius's who produce free software in their spare time that is far superior to anything even large corporations can come up with through years of dedicated development. It also means distributed and open development, consistency, compatibility and technical cutting edge information technology. GNU means doing things once in the best way possible, providing solutions instead of quick fixes, and looking exhaustively at possibilities instead of going for the most brightly coloured expedient approach.

26.2.3 Why is GNU software better than proprietary software?

Proprietary software is often looked down upon in the free software world for many reasons:

- It is closed to external scrutiny.
- Users are unable to add features to the software
- Users are unable to correct errors (bugs) in the software

The result of this is that proprietary software,

- does not confirm to good standards for information technology.
- is incompatible with other proprietary software.
- is buggy.
- cannot be fixed.
- costs far more than it is worth.
- can do anything behind your back without you knowing.
- is insecure.
- tries to be better than other proprietary software without meeting real technical needs.
- wastes a lot of time duplicating the effort of other proprietary software.
- often does not build on existing software because of licensing issues or ignorance

GNU software on the other hand is open for anyone to scrutinize it. Users can (and do) freely fix and enhance software for their own needs, then allow others the benefit of their extensions. Many developers of different expertise collaborate to find the best way of doing things. Open industry and academic standards are adhered to, to make software consistent and compatible. Collaborated effort between different developers means that code is shared and effort is not replicated. Users have close and direct contact with developers ensuring that bugs are fixed quickly and users needs are met. Because source code can be viewed by anyone, developers write code more carefully and are more inspired and more meticulous.

Another partial reason for this superiority is that GNU software is often written by people from academic institutions who are in the centre of IT research, and are most qualified to dictate software solutions. In other cases authors write software for their own use out of their own dissatisfaction for existing proprietry software - a powerful motivation.

26.2.4 Explain the restrictions of Linux's 'free' GNU General Public (GPL) software license.

The following is quoted from the GPL itself:

> When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.
>
> To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.
>
> For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

26.2.5 If Linux is free, where do companies have the right to make money off selling CD's?

See 'Where do I get Linux?' below.

26.2.6 What if Linus Torvalds decided to change the copyright on the kernel? Could he sell out to a company?

This is not possible. Because of the legal terms of the GPL, for Linux to be distributed under a different copyright would require the consent of all 200+ persons that have ever contributed to the Linux source code. These people come from such a variety of places, that such a task is logistically infeasible. Even if it did happen, new developers would probably rally in defiance and continue work on the kernel as it is. This free kernel would amass more followers and would quickly become the standard, with or without Linus.

26.2.7 What if Linus Torvalds stopped supporting Linux? What if kernel development split?

There are many kernel developers who have sufficient knowledge to do the job of Linus. Most probably a team of core developers would take over the task if Linus no longer worked on the kernel. Linux might even split into different development teams if a disagreement did break out about some programming issue. It may rejoin later on. This is a process that many GNU software packages are continually going through to no ill effect. It doesn't really matter much from the end user's perspective since GNU software by its nature always tends to gravitate towards consistancy and improvement, one way are the other. It is also doesn't matter to the end user because the end user has selected a popular Linux distribution packaged by someone who has already dealt with these issues.

26.2.8 What is Open Source vs Free vs Shareware?

Open Source is a new catch phrase that is ambiguous in meaning but is often used synonymously with Free. It sometimes refers to any proprietary vendor releasing source code to their package, even though that source code

is not 'free' in the sense of users being able to modify it and redistribute it. Sometimes it means 'public domain' software which anyone can modify, but which can be incorporated into commercial packages where later versions will be unavailable in source form.

Hence we don't like to use the term 'Open Source'. 'Free' software, in the sense of 'freedom' to modify and redistribute, is the preferred term and necessitates a copyright license along the same vein as the GPL.

Shareware refers to completely non-free software that is encouraged to be redistributed at no charge, but which requests a small fee if it happens to land on your computer. It is not free software at all.

26.3 Linux Distributions

This section covers questions that about how Linux software is packaged and distributed, and how to obtain Linux.

26.3.1 If everyone is constantly modifying the source, isn't this bad for the consumer? How is the user protected from bogus software?

You as the user are not going to download arbitrary untested software any more than you would if you were using Win95.

When you get Linux, it will be inside a standard distribution, probably on a CD. Each of these packages is selected by the distribution vendors to be a genuine and stable release of that package. This is the responsibility taken on by those who create Linux distributions.

Note that there is no 'corporate body' that overseas Linux. Everyone is on there own mission. BUT, a package will not find its way into a distribution unless someone feels that it is a useful one. For people to feel it is useful means that they have to have used it over a period of time, and in this way only good, thoroughly reviewed software gets included.

Maintainers of packages ensure that official releases are downloadable from their home pages, and will upload original versions onto well established ftp servers.

It is not the case that any person is free to modify original distributions of packages and thereby hurt the names of the maintainers of that package.

For those who are paranoid that the software that they have downloaded is not the genuine article distributed by the maintainer of that software, digital signatures can verify the packager of that software. Cases where vandals have managed to substitute a bogus package for a real one are extremely rare, and entirely preventable.

26.3.2 There are so many different Linux versions - is this not confusion and incompatibility?

(See also next question.)

The Linux kernel is now on 2.2.11 as of this writing. There are no other versions of the Linux kernel that are considered stable and suited for broad public use. The previous releases of the kernel were the 2.0.3x versions - this had been the standard for more than a year.

The Linux kernel version does not effect the Linux user. Linux programs will work regardless of the kernel version. Kernel versions speak of features, not compatibility.

Each Linux distribution has its own versioning system. RedHat has just released version 6.0 of its distribution, Caldera, 2.2, Debian, 2.1, and so forth. Each new incarnation of a distribution will have newer versions of packages contained therein, and better installation software. There may also have been subtle changes in the filesystem layout.

The Linux Unix C library implementation is called glibc. When RedHat brought out version 5.0 of its distribution, it changed to glibc from the older 'libc5' library. Because all packages require this library, this was said to introduce incompatibility. It is true however that multiple versions of libraries can coexist on the same system and hence no serious compatibility problem was ever introduced in this transition. Other vendors have since followed suite in making the transition to glibc (also known as libc6).

The Linux community has also produced a document called the Linux Filesystem Standard. Most vendors try to be compliant with this standard, and hence Linux systems will look very similar from one distribution to another.

There is hence no real confusion or compatibility problems between Linux's.

26.3.3 Will a program from one Linux Distribution run on another? How compatible are the different distributions?

The different distributions are NOT like different operating systems (compare Sun vs IRIX). They are very similar and share binary compatibility (provided that they are for the same type of processor of course) - i.e. Linux binaries compiled on one system will work on another. Utilities also exist to convert packages meant for one distribution to be installed on a different distribution. Some distributions are however created for specific hardware and hence their packages will only run on that hardware. However all software specifically written for Linux will recompile without any modifications on another Linux platform in addition to compiling with 'few' modifications on other Unix systems.

The rule is basically this: if you have three packages that you would need to get working on a different distribution, then it is trivial to make the adjustments to do this. If you have a hundred packages that you need to get working, then it becomes a problem.

26.3.4 What is the best distribution to use?

If you are an absolute beginner and don't really feel like thinking about what distribution to get, the most popular and easiest to install is RedHat. RedHat is also supported quite well in industry.

The attributes of some distributions are:

Mandrake: Mandrake is RedHat with some packages added/updated. It has recently become very popular, and may be worth using in preference to RedHat.

Debian: This is probably the most technically advanced. It is completely free and very well structured as well as standards conformant. It is slightly more difficult to install.

RedHat: The most popular. What's nice about RedHat is that almost all developers provide RedHat rpm's (the file that a RedHat package comes in). Debian deb files are usually provided, but not as often as rpm. As explained, RedHat is the easiest to install (although the new Caldera is also getting pretty cool installation wise).

Caldera: this is a commercial distribution that has an office suite built into it. Caldera is now getting as easy to install as RedHat. If you have an office that wants to put Linux on every machine, Caldera may be the way to go. You will have to order these CD's straight from Caldera though, since we are not aware of suppliers in SA.

Slackware: This was the first Linux distribution and is supposed to be the most current (software is always the latest). Its a pain to install and manage, although school kids who don't know any better love it.

SuSE: SuSE in Germany ships a very nice Linux distribution with good documentation written from scratch.

Corel: Corel corporation says that they will come out with their own Linux distribution at the end of this year. This will probably have a lot of Corel's own applications bundled in. It won't have Corel Draw though.

26.3.5 Where do I get Linux?

Once you have decided on a distribution (see previous question), you need to download that distribution or buy/borrow it on CD. Commercial distributions may contain proprietary software that you may not be allowed to install multiple times. However, Mandrake, RedHat, Debian and Slackware are all committed to freedom and hence will not have any software that is non-redistributable. Hence if you get one of these on CD, feel free to install it as many times as you like.

Note that GPL does not say that GNU software is without cost. You are allowed to charge for the service of distributing, installing and maintaining software. It is the freedom to redistribute and modify GNU software that is meant by the word free.

The international mirror for Linux distributions is

- ftp://metalab.unc.edu/pub/Linux/distributions/

You would be have a lot of free time to download from this link though, so rather use our local mirrors on ftp.is.co.za and ftp.snd.co.za. Some universities also have mirrors: ftp.wits.co.za (Wits) and ftp.sun.ac.za (Stellenbosch).

(Its a good idea to browse around all these servers to get a feel of what software is available. Also check the date of the file, since some software can really sit around for years and there may be a more recent version available elsewhere on the internet.)

Downloading from these ftp sites is going to take a long time unless you have a really fast link. Hence rather ask around who locally sells Linux on CD. Do NOT adopt the attitude that since Linux is free, one should not have to pay for a CD. CD's are very cheap (R100 or so) - money that will pay for itself many times over in the time you save.

Also always make sure you have the LATEST VERSION of whatever it is you're buying or downloading. Under no circumstance install from a distribution that has been superseded by a newer version.

26.3.6 How do I install Linux?

It helps to think more laterally when asking trying to get information about Linux:

> Would-be Linux users everywhere need to know how to install Linux. Surely the free software community has long since generated documentation to help them? Where is that documentation?

Actually, RedHat has an extremely comprehensive installation guide in html format. Browse around your RedHat CD to find it.

There is also a lot of installation guides available on the net - see what happens when you do a net search with 'linux installation guide'. Each distribution will also have an installation guide as part of it. You need to read through that guide in detail. It will explain everything you need to know about setting up partitions, dual boots, etc.

The installation procedure WILL be completely different for each distribution.

26.4 Linux Support

This section explains where to get free and commercial help with Linux.

26.4.1 Where does a person get Linux support? My bought software is supported - how does Linux compete?

Linux is supported by the community that uses Linux. With commercial systems, users are too stingy to share their knowledge because they feel that they owe nothing for having spent it on software.

Linux users on the other hand are very supportive of other Linux users. A person can get FAR BETTER SUPPORT from the internet community that they would from their commercial software vendors. Most packages have email lists where the very developers are be available for questions. Most cities have mailing lists (Gauteng and the Western Cape have ones) where responses to email questions are answered within hours. The very idea that Linux is not supported is laughed at by all those that have started using Linux. The new Linux user discovers that help abounds and that there is never want for a friendly discussion about any computing problem they may have.

Newsgroups provide assistance where Linux issues are discussed and help is given to new users - there are many such newsgroups. Using a newsgroup has the benefit of the widest possible audience.

The web is also an excellent place for support. Because users constantly interact and discuss Linux issues, 99% of the problems a user is likely to have would have already been documented or covered in mailing list archives, often obviating the need to ask anyone at all.

Finally, many professional companies will provide assistance at hourly rates comparable (usually cheaper) than for commercial software.

26.4.2 Should I buy a reference book for Linux? Where do I get one?

It is recommended that you buy both a comprehensive Unix system administrators reference as well as a Linux book. The Linux book will give the tips, tricks and Linux specifics, while the Unix reference will teach you the fundamental theory and basic practical commands of Unix.

(This is not strictly necessary - there are many online manuals tutorials, FAQ's and HOWTO's. However printed text will be a great help to those new to Unix.)

Popular book stores in South Africa are starting to stock more Unix and Linux material.

amazon.com is probably the best place to order for a wide selecting.

26.4.3 What companies support Linux in South Africa?

LPA Home Page `<http://www.lpa.org.za/>` contains a list of all companies that have joined the Linux Professionals Association (LPA).

26.4.4 What mailing lists can I subscribe to in South Africa?

Send a one line message

> subscribe clug

to the email address:

> majordomo@leg.uct.ac.za

This will subscribe you to the CLUG mailing list

> clug@leg.uct.ac.za

You will get a reply mail giving further instructions.

You can do the same for the Gauteng Linux Users Group (GLUG): majordomo@linux.org.za for glug@linux.org.za

26.5 Linux Compared to Other Systems

This section discusses the relative merits of different Unix's and NT.

26.5.1 What is the most popular Unix in the world?

It has long since been agreed that Linux has by far the highest install base of any Unix.

26.5.2 How many Linux systems are there out there?

This is a question nobody really knows. Various estimates have been put forward based on statistical considerations. 10-20 million is the current figure.

What is clear is that the number of Linux users is doubling consistently every year. This is evident from user interest and industry involvement in Linux - journal subscriptions, web hits, media attention, support requirements, software ports etc.

26.5.3 What is the TOTAL cost of installing and running NT compared to a Linux system?

Although Linux is free (or at most R100 for a CD), a good knowledge of Unix is required to install and configure a reliable server. This tends to cost you less than the same number of hours of qualified NT support, but it WILL still cost you.

On the other hand, your NT workstation has to be licensed.

Many arguments have been put forward regarding server costs that fail to take into account the completely lifetime of the server. This has resulted in contrasting reports that either claim that Linux costs nothing, or claim that it is impossible to use because of the expense of the expertise required. Neither of these extreme views are true.

The total cost of a server includes the following:

- Cost of the Operating System
- Cost of dedicated software not inherently supported by the operating system
- Cost of hardware
- Cost of installation
- Cost of support
- Implicit costs of server down-time
- Cost of maintenance
- Cost of repair
- Cost of essential upgrades
- Linux can run many services (mail, file, web) off the same server rather than having dedicated servers - this can be a tremendous saving.

When all these factors are considered, any company will probably make a truly enormous saving by choosing a Linux server over an NT server.

26.5.4 What is the TOTAL cost of installing and running a Linux system compared to a proprietary Unix system?

(See previous question.)

Proprietary Unix systems are not as user friendly as Linux. At the present time, Linux is also considered easier to maintain than any commercial Unix system (the degree depends on the system) because of its widespread use and hence easy access to Linux expertise. Linux has a far more dedicated and 'beginner-friendly' documentation project than any commercial Unix.

The upshot of this is that though your proprietary Unix system will perform as reliably as Linux, its is going to be more time consuming to maintain.

Unix's that run on specialized hardware are almost never worth what you paid for them in terms of a cost/performance ratio. That goes doubly so if you are also paying for an operating system.

26.5.5 How does Linux compare to other operating systems in performance?

Linux will typically perform 50% to 100% better than other operating systems on the same hardware. There are no commercial exceptions to this rule on the PC.

There have been a great many misguided attempts to show that Linux performs better or worse than other platforms. We have never read a completely conclusive study. Usually these studies are done with one or other

competing system having better expertise at its disposal, and are hence grossly biased. In some supposedly independent tests, Linux tended to outperform NT as a web server, file server and database server by an appreciable margin.

The recent 'Mindcraft' tests drew much controversy by showing NT to outperform Linux by a considerable margin. The scenario that was tested (including the repeat test in June 1999) is however unlikely to ever occur in practice. Even so, work has recently corrected the minor problems that caused this performance degradation. If you are one of the dozen odd companies in the world that uses a web/file server in this way, you may find that the latest kernel to be an improvement.

In general the performance improvement of a Linux box is quite visible to the user/administrator. It is especially noticeable how fast the file system access is, and how it scales smoothly when multiple services are being used simultaneously. NT may perform well when loaded by one service only, but that is not how one wants to use a server.

There is also criticism of Linux's SMP (multiprocessor) support, and lack of a journalling file system. These two issues are discussed in the next question.

In our experience (from both discussions and development), Linux's critical operations are always pedantically optimised - far more than would normally be encouraged in a commercial organisation. Hence if your hardware is not performing the absolute best it can, it's by a very small margin.

Its also probably not worth while debating these kinds of speed issues where there are so many other good reasons to prefer Linux.

26.5.6 What about SMP and a journalling file-system? Is Linux enterprise ready?

It really doesn't matter whether there are criterion to say if Linux is or isn't READY for the enterprise. The fact is that Linux is being USED in a great many close-to-mission-critical situations and in situations that require high performance, and that those administrators using it are happy with its performance. Moreover there are many situations where vendors recommend Linux over any other system for hosting their applications. The author has personally configured critical machines to the satisfaction of the client.

Linux is supposed to lack proper SMP support and therefore not be as scalable as other OS's. This is somewhat true and will prbably remain the case until December 1999 when kernel 2.4 is being released. On the other hand, one should ask how many companies are actually buying these SMP motherboards with 4, 8 and 16 processors. For 95% of the situations where the author's company had to install a server, an entry level PC was more than sufficient.

Linux also lacks a journalling file system. This ultimately means that in the event of a power failure, there is small chance that the file-system would not recover from the error, and would require manual intervention. There is a very small chance that the file-system would be damaged to the point that a package may need to be reinstalled. Neither of these two scenarios are as likely as an NT server requiring a complete re-installation, which has NEVER happened on a Linux machine.

Linux will soon have a facility of some sort that will close the whole journalling file system issue. Recent news talks of some already working code.

26.5.7 Does Linux only support 2 Gig of memory and 128 Meg of swap?

Until recently there was a 2 Gig limit, but this is no longer the case. Linux supports a full 64 Gig (sixty four gigabytes) of memory, provided that a kernel patch is applied. The author of this patch tested 8 processes running simultaneously on an 8-CPU Xeon machine having 8 Gigs of RAM. Each process used a full 1 Gig of unshared memory.

If you really need this much memory, you should be using a 64 bit system, like a DEC Alpha, or Sun UltraSparc machine.

On 64 bit systems, Linux supports more memory than most first world governments can afford to buy.

Linux supports as much swap space as you like. For technical reasons, however, your swap space must be divided into separate partitions of 128 Meg each.

26.5.8 Is UNIX not antiquated? Is its security model not outdated?

The principles underlying OS development have not changed since the concept of an OS was invented some 30+ years ago. It is really academia that develop the theoretical models for computer science – industry only implements these.

Microsoft has recently claimed that UNIX is antiquated. This would be a fair criticism if Microsoft had taken into account any of the available technology when developing their own systems. It is quite obvious that NT was developed to be somewhat compatible with Win95, and hence owes many of its limitations to the original MSDOS.

UNIX has a one-administrator, many-users security model. NT is supposed to have improved upon this: if you know of any worthwhile examples of effective use of multiple administrators under NT, please let us know.

It is interesting to question what benefit multiple-administrator-ness has to a system like NT that cannot be maintained remotely. If the administrator has to sit in front of the box to administrate it, they can just as well unplug the box and reformat its hard-disk.

For what its worth, it is in fact possible under UNIX to selectively decide what users can access what system configuration files, without giving them full administrator privileges. You can then lock the machine in a safe with just a network cable sticking out (or even a serial or parallel port cable) and administer the whole thing through that.

As regards security: a general principle for any mechanical or electronic system is that something cannot be said to be secure before it is at least functionally reliable.

26.5.9 What is C2 certification? Windows NT has it, why doesn't Linux have it?

C2 basically means that a user can specify which other users can access his/her files. It also says that a log of who accessed what must be kept.

It has nothing to do with how difficult it is to compromise a system or to underhandedly gain administration privileges.

Here is the EXECUTIVE SUMMARY of the National Computer Security Centre (NCSC) Final Evaluation. (Incidently, this executive summary was compiled using LaTeX - a typesetting language for UNIX.)

> The security protection provided by Microsoft Windows NT Workstation and Server Version 3.5 with Service Pack 3 when configured according to the Windows NT Trusted Facility Manual, has been evaluated by the National Security Agency (NSA) The security features of Microsoft Windows NT Workstation and Server were examined against the requirements specified by the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC) dated December 1985 to establish a rating. The evaluation team has determined that the highest class for which Microsoft Windows NT Workstation and Server Version 3.5 with Service Pack 3 satisfies all the specified requirements of the TCSEC is C2. In addition, the B2 Trusted Path and B2 Trusted Facility Management functional requirements are also satisfied. A system that has been rated C2 provides a Trusted Computer Base (TCB) that enforces a Discretionary Access Control (DAC) policy to protect information and allow users to share information under their control with other specified users, identification and authentication of users to control access to the system and enforce accountability, the prevention of access to residual information from a previous user's actions, and the auditing of security related events. A system that satisfies the B2 Trusted Path requirement supports a trusted communication path between the TCB and the user for identification and authentication. A system that satisfies the B2 Trusted Facility Management requirement supports the ability to separate operator and administrator functions. Microsoft Windows NT Workstation and Server Version 3.5 with Service Pack 3 is a preemptive multitasking multiprocessor operating system which runs on both CISC (Intel Pentium) and RISC (DEC Alpha) hardware architectures. The platforms in the evaluated configuration are: the Compaq Proliant 2000 and 4000 and the DECpc AXP/150.

On page 3 of the report it is clearly stated that the "(POSIX) component is eliminated (i.e. not considered for review) because its UNIX-based security features have not been integrated well with the security features of the Windows NT operating system." and also that "The networking components of Windows NT are eliminated in this configuration."

In other words, the C2 security of NT would mean practically nothing to any corporation anywhere.

The NCSC is entirely a subsidiary of the National Security Agency (NSA) of the US government. Since the US government prohibits export of encryption technology, and since a secure facility requires encryption technology to have any hope of being useful in practice, the TCSEC is of no use to anyone outside of the US. The TCSEC's aim (to provide industry with a benchmark) implicitly refers to products distributed in the US only. I.e. not Microsoft.

Linux will never be C2 certified because the certification requirement dictates that (page 2) "the release being evaluated must not undergo any additional development" (an obvious requirement for a package). Which in the case of Linux is impossible. It may however be possible to certify a particular binary version of Linux, but then that version would become useless if any security exploits were ever discovered. In the first place, Linux does not have C2 features (AFAIK).

It is questionable whether the TCSEC will ever have any relevance to mainstream IT infrastructures. The Internet Engineering Task Force (IETF) is really the organisation that dictates security protocols, which is closer to what we are really talking about when we say "security".

26.5.10 What are claimed to be the principle differences between Unix and NT that make Unix advocates against NT?

Many claims are frequently made against NT by Unix advocates. These sometimes spring from rumors that circulate about the internals of the NT system as well as about politics surrounding its development. Unix advocates will frequently jeer at what they think are poorly implemented protocols, expedient subsystems, bad design decisions or insecure services. In particular, many stories circulate about how NT failed at a particular function where a Unix box performed admirably. They speak of experiences where NT crashed unpredictable or under particular loading conditions, and theorize about how smaller cheaper systems running Linux/FreeBSD/etc could outperform NT. Some of the arguments comparing Unix to NT require a considerable technical understanding to appreciate, so will not be explained here. The following broad outline gives some idea of the Unix advocate's position.

Unix conforms to many tried and tested standards such as POSIX, BSD and SYSV. These standards dictate the Application Programmers Interfaces (API), file system layout, communication protocols and data formats underlying an OS. In the case of NT these were created mostly from scratch thus inventing its own standard. No external review process existed to ensure that the created standards were enduring and technically sound. Unix, on the other hand, was designed to be a standard to which many different implementations could conform and has evolved over a long period of time.

NT is not a true multi-user system. It is sometimes thought that NT is multi-user because of its ability to host, for example, multiple simultaneous ftp logins. Such does not constitute true multi-user-ability. A Unix system allows multiple users to log onto the same server and all start running spread sheets, word processors and web browsers, completely unaware of each other's presence on the system, and each with their own secure environment that other users cannot access.

Some may argue what the point of such a flexible system is, since in most cases, each user has their own stand-alone PC, and an internet server has no-one logged onto it at all. The point is many fold: Multi-user-ability isolates separate subsystems so that they cannot interfere with each other. It provides a stable and generic framework within which services can operate. It secures individual services from compromising a system's stability and security.

The administrator and developer capitalize on this framework, to more easily and soundly develop, customize, and support software. New Unix users very soon can't live without this flexibility.

Another major difference between Unix and NT is the apparent target market of administrators who are likely to use these systems. NT does not require the administrator to understand software programs. Unix on the other hand is widely driven by scripting languages for its configuration and maintenance: the administrator is also a developer, and need not rely on the existence of a pre-written package to solve a problem - the feature can be created, and there are always many ways to do it.

Hence, on Unix, the administrator's creativity and ingenuity allow them to quickly and easily customize the system. Unix programs are in general designed to inter-operate with each other allowing the automation of any program within a scripting language. Nationwide database networks have been easily created through simple scripts, without the need for expensive development or purchasing of software. A simple example is often sited by

Linux advocates: 'Try add a new user on NT. Now try add a thousand new users'. Its very easy to do something with mouse clicks. Its far more important to be able to automate a process.

Unix systems are designed to be network transparent at their lowest levels. This means that whatever a program does, it should be able to do across a network. Whenever you use a Unix machine you are using it through a logical or physical network layer, thus facilitating any kind of remote access to a server. This means for example that you can do remote administration. You do not have to have a program installed on your machine to run it, or have a file on your machine to access it. Anything that you can do with your computer you can do on another computer across a network with complete transparency to the fact. Some feel that NT does not exploit the full potential of networking.

In terms of down-time, stability and performance, Unix systems are conjectored to be far better than NT. A Unix system will stand for years without a reboot - any configuration changes made will necessitate that only the affected subsystem be restarted and not the entire machine. Unix systems on PC's are reported to be more scalable and faster than NT.

As a developer, Unix code writers have countless GNU packages available to them as examples, whereas traditional industry does not disclose its source code. The Unix developer can hence freely borrow and review code of other packages thus making development much more economical. Unix development tools have evolved over a much longer period of time than NT, and are written and supported by the very community that uses them.

26.5.11 What do Linux users say when they compare SCOs Unix to Linux? Should I upgrade to Linux from SCOs Unix?

Some Linux users that use SCOs Unix have claimed that it is inflexible and difficult to maintain. It does run the X Window System, but generally the utilities that come with a basic machine have far less options and features compared to the GNU utilities which Linux users are used to.

It is, on the other hand, an extremely reliable server that has security features not present in Linux. Unfortunately, some Linux users consider these and many of the other distinguishing features of SCOs Unix to be inapplicable to them.

Because Linux flawlessly runs native binaries for this system, it might be a good idea to upgrade your machine to Linux. You can still keep any third party software without having to get Linux versions of them, and then have the benefit of many features that SCOs Unix would require licenses for.

26.5.12 What do Linux users say when they compare Solaris/SunOS to Linux? Should I switch to Linux?

Suns systems are of the best around. Like other Unix's, Solaris is stable as a rock. On older SUN workstations Linux has been shown to substantially improve performance over the original OS, so Linux has been installed to breath new life back into this hardware.

On the latest high end SUN servers, we are not sure if its worth reinstalling a different operating system. However its an excellent idea to install a full set of GNU utilities to make administration and development easier.

If you do install, say, RedHat on a SUN, you would have the benefit of RedHat package management and a nice array of useful programs that would take a long time to install under the native OS. The real advantage would be if you had other Linux systems on the network - your SUN box would then look exactly the same and hence be easy to administer.

Solaris for the PC is available, but is said to be slower than Linux.

26.5.13 How does FreeBSD compare to Linux?

FreeBSD is very similar to Linux in that it also relies on a large number of GNU packages. Most of the packages available for Linux are also available for FreeBSD.

The arguments comparing FreeBSD to Linux center around the differences between how various kernel functions are implemented. Depending on the area you look at, either Linux or FreeBSD will have a better imple-

mentation. On the whole, FreeBSD is thought to have a better architecture, although Linux has had the benefit of being ported to many platforms and has a great many more features and supports far more hardware. It is questionable whether the performance penalties we are talking about are of real concern in most practical situations.

GPL advocates take issue with FreeBSD because its licensing allows a commercial organisation to use FreeBSD without disclosing the source code.

None of this offsets the fact that either of these systems are preferable to proprietary ones.

26.5.14 Should I switch to Linux from IRIX?

IRIX is also (see previous questions) a good system. You should not try replacing IRIX with Linux unless you are a really keen experimenter. Linux support for the SGI is not as good as for the SUN.

Do however download all the GNU utilities you can get your hands on. SGI's are even better machines once given the GNU touch.

Its worth mentioning that Silicon Graphics has actually pledged tremendous support to Linux. Silicon Graphics Intel based workstation are being shipped that run only Linux. Silicon Graphics is said to be in the process of scrapping its NT support. Because Silicon Graphics seems to be porting important kernel features from IRIX to Linux, one can hypothesise that they are considering converting everything over to Linux in the future.

26.6 Technical

This section covers various specific and technical questions.

26.6.1 Are Linux CD's readable from Win95/98/00 ?

Yes. This will allow you to browse the installation documentation on the CD.

26.6.2 Can I run Linux and Win95 on the same machine?

Yes, Linux will occupy two or more partitions, while Win95 will sit in one of the primary partitions. At boot time, a boot prompt will ask you to select which operating system you would like to boot into.

26.6.3 How much space do I need to install Linux?

A useful distribution of packages that includes the X Window System (Unix's graphical environment) will occupy less than 1 gigabyte. A network server that does not have to run X can get away with about 300 megabytes. Linux can run on as little as a single stiffy disk - thats 1.4 megabytes - and still perform various network services.

26.6.4 What are the hardware requirements?

Linux runs on many different hardware platforms, as explained above. The typically user should purchase an entry level PC with at least 16 megabytes of RAM if they are going to run the X Window System smoothly (Unix's graphical environment).

A good Linux machine is a PII 300 (or AMD, K6, Cyrix etc.) with 64 megabytes of RAM and a 2 megabyte graphics card (i.e. capable of run 1024x768 screen resolution in 15/16 bit color). 1 gigabyte of free disk space is necessary.

If you are using scrap hardware, an adequite machine for the X Window System should not have less than a 486-100MHz processor and 8 megabytes of RAM. Network servers can run on a 386 with 4 megabytes of RAM, and a 200 megabyte hard drive.

Note that recently some distributions are coming out with Pentium only compilations. This means that your old 386 will no longer work. You will then have to compile your own kernel for the processor you are using, and possibly recompile packages.

26.6.5 What hardware is supported? Will my sound/graphics/network card work?

About 90% of all hardware available for the PC is supported under Linux. In general, well established brand names will always work, these will tend to cost more though. New graphics/network cards are always being released onto the market, If you buy one of these, you may have to wait many months before support becomes available (if ever).

To check on hardware support check the Linux hardware howto:

- *Hardware-HOWTO* `<http://users.bart.nl/~patrickr/hardware-howto/Hardware-HOWTO.html>`

This may not be up to date, so its best to go to the various references listed in this document and get the latest info.

26.6.6 Can I view my Win95/98/00/NT, DOS, etc. files under Linux?

Linux has read and write support for all these file systems. Hence your other partitions will be readable from Linux. In addition, Linux has support for a wide range of other file systems like those of OS2, Amiga and other Unix systems.

26.6.7 Can I run DOS programs under Linux?

Linux contains a highly advanced DOS emulator. It will run almost any 16 or 32 bit DOS application. It runs a great number of 32 bit DOS games as well.

The DOS emulator package for Linux is called dosemu. It will typically run applications much faster than normal DOS because of Linux's faster file system access and system calls.

It can run in an X window just like a dos window under Win95.

26.6.8 Can I recompile Win95/98/00 programs under Linux?

Yes. WineLib is a part of the Wine package (see below) and allows Windows applications to be recompiled to work under Linux. Apparently this works extremely well with virtually no changes to the source code being necessary.

26.6.9 Can I run Win95/98/00 programs under Linux?

Yes and no.

There are commercial emulators that will run a virtual 386 machine under Linux. This enables mostly flawless running of Win95/98/00 under Linux if you really have to and at a large performance penalty. You still have to buy Win95 though and your Win95 applications will not be able to directly communicate with your Linux applications.

There is also a project called Wine (WINdows Emulator) which aims to provide a free alternative to Win95 by allowing Linux to run Win95 16 or 32 bit binaries with little to no performance penalty. It has been in development for many years now, and has reached the point where many simple programs work quite flawlessly under Linux.

Get a grip on what this means: you can run Minesweep under Linux and it will come up on your X Window screen next to your other Linux applications and look EXACTLY like what it does under Win95 - all this without having to buy Win95. You will be able to cut and paste between Win95 apps and Linux apps.

However, many applications (especially large and complex ones) do not display correctly under Linux, and/or crash during operation.

Many Win95 games do however work quite well under Linux.

We personally do not find a need to use any Win95 software on Linux, having become so accustomed to native Unix software. If there is an application that you really HAVE to get working under Linux, perhaps you can try Wine on it - it may work adequately for your needs.

See the *Wine Headquarters* <http://www.winehq.com/faq.html> for more info.

26.6.10 I have heard that Linux does not suffer from virus attacks. Is it true that there is no threat of viruses with Unix systems?

Yes. There is categorically NO such thing as a virus under Linux. It has been rumored that viruses (worms) have existed in the past for Unix and have propagated themselves through networks. We have never seen such a thing and it is doubtful that anything like this managed to compromise data.

The reason a virus cannot exist on a mult-user system is because any program that runs on such a system, does so inside a protected 'privilege' space. It can only have access to the files and hardware that has been previously been allowed by an authentication process. It is hence a technical impossibility for the kinds of viruses that exist for other systems to propagate on multi-user systems like Unix.

The problem that a virus has is analogous to a real virus attacking a tobacco mono-culture. It propagates much more quickly and easily when all the hosts are the same. Unix systems are mostly not compatible AT THE LOW LEVEL WHERE A VIRUS WOULD NEED TO WORK, even different versions of the same system. Hence writing a virus that would work properly on a large variety of systems is next to impossible. Note that this has nothing to do with the 'compatability' from a users perspective.

However, although Linux cannot itself execute a virus, it may be able to pass on a virus meant for a Windows machine should a Linux box act as a mail or file server. To avoid this problem, numerous virus detection software for Linux is now becoming available.

26.6.11 Is Linux Y2K compliant?

Yes. The Linux kernel itself has no limitations as far as the turn of the millennia is concerned, since Unix dates are always calculated in seconds since 1970-01-01 and not using a format that, for example, omits the century digits.

All software that we have encountered for Linux uses the same or better date representation. Most large critical packages will have a Y2K statement on their web page.

BIOS's may however contain Y2K limitations, but then this would have nothing to do with Linux per se.

It is of course difficult to vouch that an antiquated specialized piece of software will not suffer Y2K limitations - that must be found out from the authors of that software, or through testing procedures.

In our testing, all the critical packages perform seamlessly through the turn of the millenia.

More information can be found at:

- *Linux Y2K Information* <http://www.linux.org/help/beginner/year2000.html>

26.6.12 Is Linux as secure as other servers?

Linux is as or more secure than typical Unix systems.

There are various issues that make it more and less secure:

Because GNU software is open source, any hacker can easily research the internal workings of critical system services.

On the one hand, they may find a flaw in these internals that can be indirectly exploited to compromised the security of a server. In this way, Linux is LESS secure because security holes can be discovered by arbitrary individuals.

On the other hand, they may find a flaw in these internals that they can report to the authors of that package, who will quickly (sometimes within hours) correct the insecurity and release a new version on the internet. This makes Linux MORE secure because security holes are discovered and reported by a wide network of programmers.

It is therefore questionable whether free software is more secure or not. We personally prefer to have access to the source code so that we know what our software is doing.

Another issue is that Linux servers are often installed by lazy people who do not take the time to follow the simplest of security guidelines, even though these guidelines are widely available and easy to follow. Such systems are sitting ducks and are often attacked.

A further issue is that when a security hole is discovered, system administrators fail to head the warnings announced to the Linux community. By not upgrading that service, they compromise security.

It is possible to make a Linux system completely air tight by following a few simple guidelines, like being careful about what system services you expose, not allowing passwords to be compromised etc.

Because of the community nature of Linux users, there is openness and honesty with regard to security issues. It is not found, for instance, that security holes are covered up by maintainers for commercial reasons. In this way you can trust Linux far more than commercial institutions that think they have a lot to loose by disclosing flaws in their software.

26.7 Software

This section covers questions about what software is available for Linux, and about Linux's graphical user interface.

26.7.1 What office suites are there for Linux?

StarOffice is an outstanding suite available for many Unix's and also for Win95. It has a large proportion of the market share in the country where it was developed, Germany. It is now available for free off the internet, or can be ordered for $10 from Sun Microsystems who has recently purchased StarDivision.

Applixware is another comparable office suite.

Corel's entire Office suite will be available for Linux in the year 2000.

KOffice is a new freeware office suite, and the Gnome project also has a powerful GUI spreadsheat application.

In addition to these, there are many free word processors and spread-sheats out their. Most will not be as user friendly or as powerful as commercial office suites, but are certainly speedier and more stable.

26.7.2 What is the best way to do professional typesetting on Linux?

You might be surprised that a WYSIWYG word processor is not the best way to produce professional documents. Under Linux there is a typesetting language called LaTeX which is used to produce most scientific and mathematical journal articles in academia.

Complete online documentation for LaTeX installs by default with most popular Linux distributions. There are also many tutorial books on the subject.

The typeset version of this document is produced using LaTeX.

26.7.3 What is the X Window System? Is there a graphical user interface for Unix?

(This is somewhat of a technical description - the user need not be intimidated by any of this to get their graphical environment running on Linux. To learn about different Linux desktop environments, see 'What is GNOME/KDE?' below.)

Unix has long since had a standard graphical environment called the X Window System System [Version 11] (X11 or X for short). The X Window System is a large and complicated system for generating graphics (windows) to create the beautiful user interfaces that we are accustomed to. Linux has its own port of X called XFree86.

Most Unix systems run X. All of the Unix systems mentioned above run X. Hence Linux programs will recompile for any of these systems with few or no changes and vice-versa.

X is a system of interfacing a program with the graphics that it would like to display. On the one hand sits the program, and on the other sits an X 'server'. This X server contains the necessary drivers for the particular graphics hardware (card). The two communicate via a network socket. This means that if a computer is on a network, the user can run any X application and cause it to display on the screen of any other machine. This is why X is called a 'network transparent' windowing system.

X allows tremendous flexibility in an office environment. Software need not be reinstalled on every machine because it can be run from anywhere in the office. Thin clients are possible and users can work on different machines without getting up from their desks - even if those machines run different hardware or use a different Unix. Because X is such a well designed system, the performance penalty for doing this is not noticeable on 10 megabit ethernet - i.e. a typical LAN.

X itself, however, only allows primitive drawing commands - lines, text, circles. What gives Linux its beautiful graphical interface is the window manager (what allows windows to be resized and minimized, and what would display the 'start' button and icons) and the widget libraries (see Gtk/Qt below) which produce complex graphics from these drawing primitives. Because these can be chosen by the user, and are not an inherent part of X, your Linux system can look like any of Win95, MacOS, Nextstep, or OS2. In this way graphics are far more flexible than non-X operating systems.

It has been claimed by the uninformed that X is over-engineered and cumbersome. Indeed this is complete nonsense - the X Window System in an ingenious standard that solved the problem of graphics hardware incompatibility as well as network transparency. The computing industry could scarcely have asked for a better standard.

26.7.4 What is Gtk?

Old Unix used to have horrible looking widget libraries (the GUI (Graphical User Interface) programmers libraries used to draw buttons, menus etc.)

Sun came out with Xview - which was nicer looking but still quite hideous. A library called Motif was also developed which was extremely powerful, good looking, but also very slow. Motif is still a proprietary library which has to be purchased.

Various other widget libraries were also developed by free and proprietary groups. None met the demands of a high class GUI that could become the universal free standard.

Gtk was then developed which was later improved and made object oriented in its current incarnation: Gtk+. It is written in C.

Gtk is a programmers widget library that anyone can use to develop beautiful graphical applications under the X Window System using C or C++. It is extremely clean, efficient and well implemented and is also very fast.

Gtk consists of three libraries:

- libglib - this is a set of C utility functions to extend the standard C library calls.
- libgdk - this is a wrapper around low level X Window System calls providing GNU naming conventions for functions and implementing a simpler way to access X's functionality.
- libgtk - uses gdk to implement higher level widgets like buttons, check boxes, entry widgets, menus, scrollbars etc.

26.7.5 What is Gnome?

Gnome is an ergonomic and aesthetically pleasing desktop environment for Unix built with Gtk. It aims to achieve a consistent look and feel for all X applications and bring greater user-friendliness and beauty to the

Unix screen. Version 1 of Gnome has recently been release and sports the utilities, look and feel expected of a modern desktop environment.

Gnome is also a higher level programmers interface to Gtk, used for writing applications that conform to the Gnome style guides (see *Gnome Desktop* <http://www.gnome.org/>).

Gnome also implements a remote procedure call mechanism where applications can communicate with each other in a network transparent way. It uses Orbit - a free CORBA implementation for this.

26.7.6 What is Qt?

Qt is C++ widget library that is available for Unix as well as Win95 and Mac systems. It was originally proprietary but is now Open Source although it is not under the GPL. It serves the same purpose as Gtk.

26.7.7 What is KDE?

KDE is to Qt as Gnome is to Gtk. The KDE project started before Gnome and is hence more advanced than Gnome. It is mostly (all?) C++ based. (see *KDE Desktop* <http://www.kde.org/>)

26.7.8 What is Gimp?

Gimp stands for GNU Image Manipulation Program. Gimp is a tool for photo retouching, image composition and image authoring. It is a general tool for manipulating raster type images like the imaging tools by Adobe.

Gimp is a state of the art piece of software. It was the first GPL desktop application to truly out-power even the most expensive commercial equivalents. Anyone doing artwork or image editing should get their hands on Gimp.

Gimp and Gtk were written in parallel. Gimp uses the Gtk library.

26.7.9 What media players, image viewers, mail/irc/news clients, and web browsers are available for Linux?

Linux has a lot of free software programs performing all these functions. Netscape navigator/communicator is available for Linux and is packaged with most distributions. The new Netscape 5.0 will be available soon and promises to be faster and more powerful than any other browser.

There is an abundance of free sound players and image viewers/editors - some of these will exceed the expectations of Win95 users. To play video files will sometimes require proprietary modules that are not available for Linux. However most common video formats can be viewed.

In many cases, the software you will find will not be as glitsy as its Win95 equivalent. In particular, a mail client as powerful as Outlook Express is not available, although there are many very powerful text only mail clients, and some up and coming graphical ones.

26.7.10 Can I use Visual Basic (VB) under Linux? What Unix alternatives are there?

No. Visual Basic is a purely Microsoft standard and is not available for Unix. VB will probably never be widely used in the Unix community where there are thought to be far better object oriented languages available.

There is however one product to convert VB code into java and and run active server pages: *Halcyon Software* <http://www.vbix.com/>.

Python, for example, is an object oriented scripting language said to far exceed VB. Anyone wanting to do Rapid Application Development (RAD) should look at the *Python Home page* <http://www.python.org/>. and download the Gnome and database modules for python.

26.7.11 Can I run Active Server Pages under Linux?

See *Halcyon Software* <http://www.vbix.com/> for a product to run Active Server Pages under Unix.

Note that, once again, there are better utilities for creating interactive web pages under Linux. PHP, Perl and Python are typically used for this.

26.7.12 Can I develop with Java under Linux?

Yes - Java development utilities and libraries are well supported under Linux. Linux has an extremely fast virtual machine called kaffe, and there is a large support community for Java development under Linux.

IBM have released opensource 1.1 JDK for Linux, including superb best-of-breed Jikes compiler and their own high-performance 1.1 JVM for Linux.

Linux's native C compiler now supports a Java frontend, allowing Java 1.1 programs to be compiled natively on Linux (currently AWT is not supported, but hopefully a GTK-based AWT implementation will be available soon).

Sun's 1.2 JDK is available for Linux, but not yet stable.

Borland has released JBuilder 3 for Linux, but being in pure Java, is very slow.

Some would recommend against programming in Java. There are already far more well established object oriented scripting languages that are more powerful than Java and are freely available. You should in particular investigate using Python before trying Java. Python is thought to offer a much simpler and more intuitive syntax, affording far more rapid development. Python can also be byte compiled (into its own native bytes or even into Java bytes) and runs on as many different platforms. Its syntax also doesn't suffer from the tedium that Java inhereted from the C++ compiler language.

26.7.13 How do I develop platform independent programs under Linux?

With Python and wxWindows. See the *Python Home page* <http://www.python.org/> for more details. For those who want to write a GUI program that runs under both Win95 and Unix, Python is BY FAR the best way to do this. Do not try to do this with Java or Tcl. These are both redundant technologies.

26.7.14 Can I develop using C/C++ under Linux?

Yes. GNU in particular encourages C development because of its compatibility across other Unix's. Unix (and in fact all operating systems) are entirely structured around the C language. C++ is also fully supported.

Note that these development environments are an inherent part of the Unix system. You will not have to separately install them and they will come with all Linux distributions as a matter of course.

26.7.15 What Integrated Development Environments (IDE's) are available for Linux? How do I develop my own applications?

Under Unix, all packages integrate with each other to provide seamless pooling of functionality. An IDE of the kind used under Win95 is not recommended even though there are some available.

How Linux developers program is using a programmers text editor. Unix text editors have far more functionality than typical IDE's available for Win95. The user has more control and flexibility than with a monolithic IDE.

One of the many free text editors should be tried before going to a commercial development environment, and will greater reduce your development time, once an initial learning curve is transcended. Also remember that commercial development environments do not guarantee enduring support, whereas freeware languages stick around for decades.

Your editor will interface with a variety of programs - the C/C++ compiler, linker, debugger, code syntax tools, library and code organization tools - to provide what effectively behaves as an IDE.

Using this process, development under Unix is sometimes thought to be a far faster and smoother process than under Win95.

If you really want an IDE, there is Cygnus's a CodeFusion, KDE's KDevelop, Gnome's gIDE, and Metrowerks' Code Warrior.

Borland has announced that they will be porting Delphi and their C/C++ Builder to Linux.

26.7.16 What other development languages are available for Linux? What is typically being used?

Perl is a well know scripting language used by administrators that is also excellent for CGI scripting. Its the recommended language for writing non-interactive administrative tools.

Python (see above) is an object oriented scripting language that can be used to to build GUI programs. It is the recommended language to build any type of interactive or GUI program. Python can be used to build platform-independent GUI applications across Win95 and Unix using wxPython.

Tcl is a powerful scripting language that has been technically superseded by Python. It used to be the only scripting language that could be used to build platform-independent GUI programs across Win95, Mac's, Unix's and OS2, and may still do this better than any other language.

C/C++ are mentioned above.

Objective-C is another C variant that is supported.

Shell scripting is the 'DOS Batch' programming of Unix. It is comprehensive enough that full GUI applications can be written with it given some effort. Unix initialization, configuration and administration are all driven by shell scripts, and this is where they see their greatest usage.

Lisp (also known as Scheme) is possibly the most expressive language available and one of the oldest interpreted languages. It was used to develop expert systems (artificial intelligence) and now sees wide use in making applications that are extensible by giving them a builtin scheme interpretor. It is very popular in the free software community.

A Pascal to C compiler is available for Linux. It can be used in this fashion as a Pascal development tool, and for porting existing applications. It is recommended against writing new applications in Pascal, however.

Borland is in the process (?) of porting their development tools to Linux. I wouldn't bother with these if you are developing a new application.

Java is mentioned above.

There a great many other languages available for Linux often developed as free alternatives to their commercial counterparts. Do a net search to see if an interpretor/compiler is available for your language under Linux.

26.7.17 Are there SQL servers available for Linux? Are there free SQL servers?

Most of the commercial SQL servers are available for Linux, and certainly all of the better ones.

There is however an excellent freeware SQL server called Postgres, that obviates the need to pay the expensive licensing fees of commercial servers. Postgres does not have all the features of some commercial systems and is not as fast, but is adequate in 99% of all cases. Postgres also has a number of features that are not available for other SQL servers that Postgres developers soon find are indispensable.

There are also other shareware SQL servers.

26.7.18 Is there a SWAN (Secure Wide Area Network) available for Linux?

Yes. A complete SWAN implementation called FreeSWAN is available under the GPL that implements the IETF (Internet Engineering Task Force) SWAN protocols.

This is all you want as far as SWAN technology goes.

26.8 Microsoft Issues

26.8.1 What is the story with Microsoft being ruled a monopoly?

Microsoft was recently ruled to be a monopoly in a lawsuit between them and the justice department.

Microsoft has rebutted against the arguments of the court (why I dunno — since the decision has already been made.)

I have become aware that the majority of people really don't have any idea what this law suite is about (I certainly didn't until I had read the rebuttal.)

Most people see the question 'is bundling IE with MSW illegal?' and the question 'is MS a monopoly?'. These two seem rather stupid questions, hence the wide controversy behind the case.

However, this is the media's projection of the case. In fact the court case has nothing to do with such questions.

The US justice department is trying to show that Microsoft's has acted in violation of the Sherman act — some business precedent in the US. Their violation is rather complicated: the phrase 'bundling IE with MSW' is the summary of a rather complicated legal argument.

The court case really has to do with Microsoft weilding too much power; that Microsoft has to be brought down, and that there are no laws that clearly say that Microsoft is doing anything 'wrong'.

On the other hand, Microsoft really has done some things which are flatly unethical. See Caldera's law suite for example (on their web site). Whether or not these are violations of specific laws is not a straight forward question.

I think behind the scenes the department of justice is bent on bringing Microsoft down. They are doing it through whatever means are necessary, including any kind of ridiculous argument they can come up with. The whole court fiasco is just a front and it may backfire on them completely. I hope it does, because its not Microsoft that is primarily at fault here.

Microsoft is a monopoly for the following reasons:

1. Microsoft grossly violated the Sherman act by selling ANY products that could not be bought WITHOUT ALSO BUYING WINDOWS 9x. The court NEVER mentions this because its an act of which every other software house in the world is also guilty! The Sherman act states "...an agreement by a party to sell one product but only on the condition that the buyer also purchases a different (or tied) product, or at least agrees that he will not purchase that product from any other supplier." The term 'agreement' should extend to 'requirement' in the case of software, provided the functionality of the one product is considered reasonably separate from that of another, as is the case with word processors, spread sheets etc. compared to the operating system that they run on. This extends the obvious intention of "...an agreement..." to apply to software. This is the single overriding reason why Microsoft is a monopoly. They pursued these habits unchecked for the entire of their growth, and no-one said anything. This precisely where the law needs to be extended, simply because it is far too easy to make software that forces people to buy other software.

Consider the analogy in the car industry. Say BMW manufactured a 4x4. Then they manufactured a trailer that the 4x4 could tow. The 4x4 however has a special hook that only a BMW trailer can fit. However, BMW has a patent on that hook, so that no one can manufacture a trailer that can be towed by a BMW or a car that can tow a BMW trailer. It would be commercial suicide for BMW to place such a limitation on their products, where they have only a few percent of the market share on trailers and cars. However, lets say that they had 80% of the market share and implemented this scheme. Now would you consider BMW's tactics unfair? Even if they aren't unfair per se, they are certainly bad for the industry. You can see that the scenario is totally unrealistic from an engineering point of view — PHYSICAL products have their own limitations which prevent companies from becoming monopolies. Not so with software products.

2. Microsoft has a superb marketing strategy.

3. Microsoft is not just a software company. It is a massive money pyramid scheme that uses loopholes in financial indicators to appear to be worth ten times its actual value.

4. Microsoft is a monopoly because of the secrecy of its protocols, API's and file formats.

5. Microsoft is a monopoly because it really DOES produce SOME excellent software.

6. Bill Gates really is a nice guy; albeit a little misguided; I would sooner him running the world than a lot of other CEOs.

7. LAST AND LEAST, Microsoft is a monopoly because of unethical business practices.