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

H4.SMR/986-10

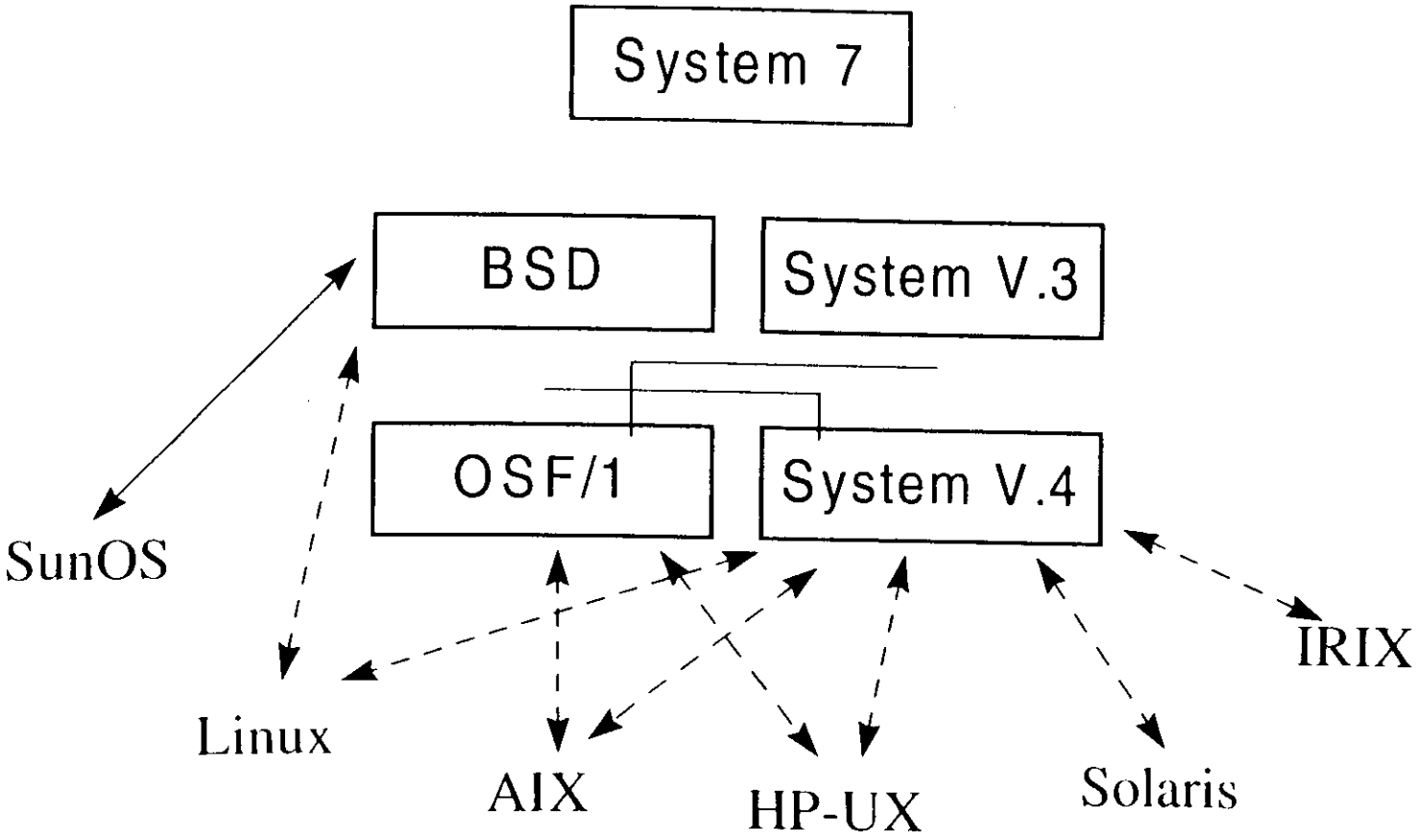
ICTP - URSI - ITU/BDT WORKSHOP ON THE USE OF RADIO FOR DIGITAL COMMUNICATIONS IN DEVELOPING COUNTRIES

(17 - 28 February, 1997)

Introduction to LINUX

E. Pietrosemoli

University of Merida
Merida
VENEZUELA


System 7
BSD
System V.3
OSF/1
System V.4
SunOS
Linux
AIX
HP-UX
Solaris
IRIX

UNIX characteristics

UNIX is a virtual machine

UNIX is a complete system

Written in C

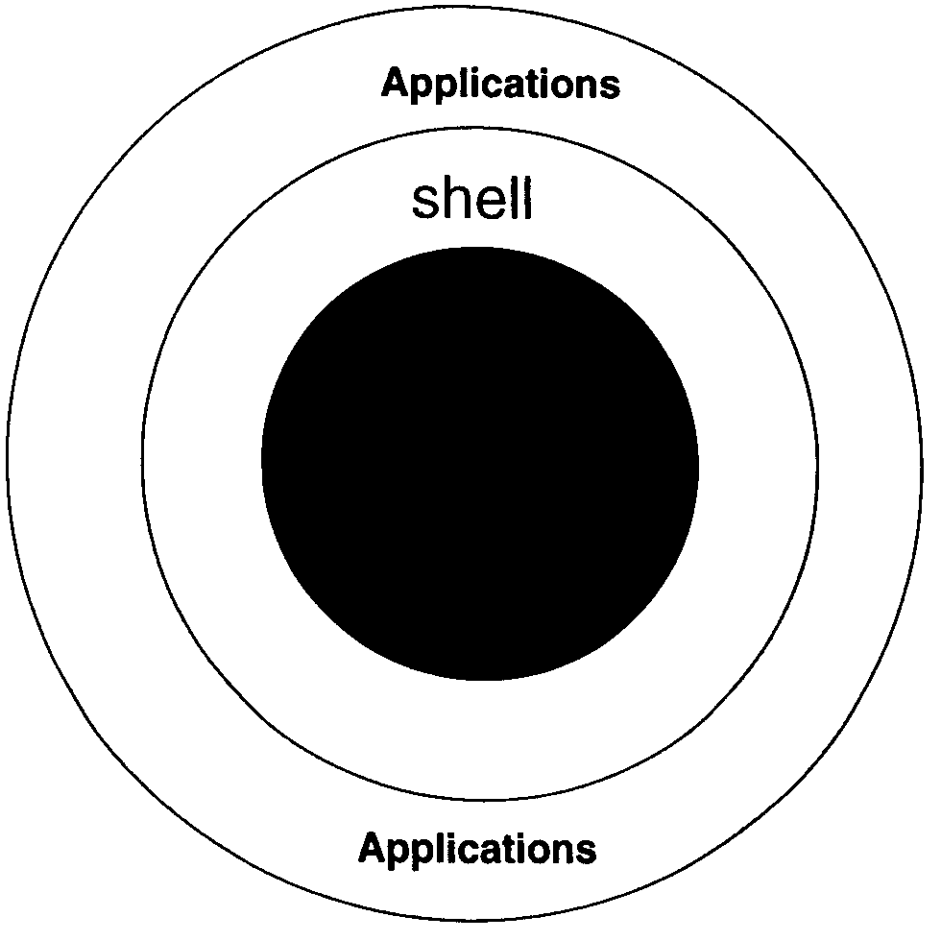
Designed for programmers

Multi-user

Multitasking

Time sharing

System structure



System access and exit

Computer Lab - Welcome

Login: user1.

Password: ****

:

:

:

$ Logout

Changing the password

```
$ passwd
Old Password: ******
New Password: ******
Re - Enter New Password: ******
```

Passwords Dos and Don'ts

Don't use a real word or name - Make up a word, be creative!

Don't use dates - Dates can be easily guessed

Don't write it down - Passwords must be easy to remember

Don't give your passwords to others

Do include upper and lower case letters, numbers and punctuation

Do make the password more than 6 character long (up to 8 on most systems)

Do use different passwords on different machines

Do change passwords often

User environment

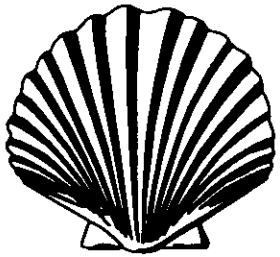
Main environment variables:

* HOME
* PATH

Main configuration files:

* .cshrc
* .profile
* .login
* .mailrc

The shells

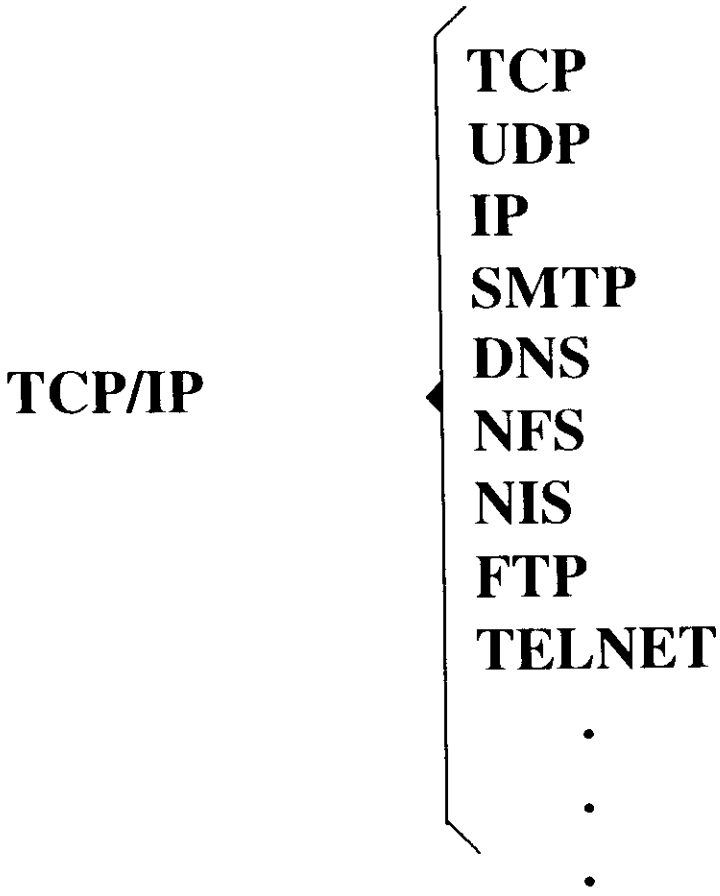


- *BOURNE SHELL - (sh)*
- *C - SHELL - (csh)*
- *BOURNE AGAIN SHELL - (bash)*
- *KORN SHELL - (ksh)*

Shell's functions

- Program execution (;).
- Name substitution and gobbling (*, ?).
- **Input/output redirection and pipes (|, > , >>, <).**
- Environment control
- Programming language

Networking with UNIX



Working with files

ls

rm graphic1

lpr chapter1

cp pict1 pictures/pict1

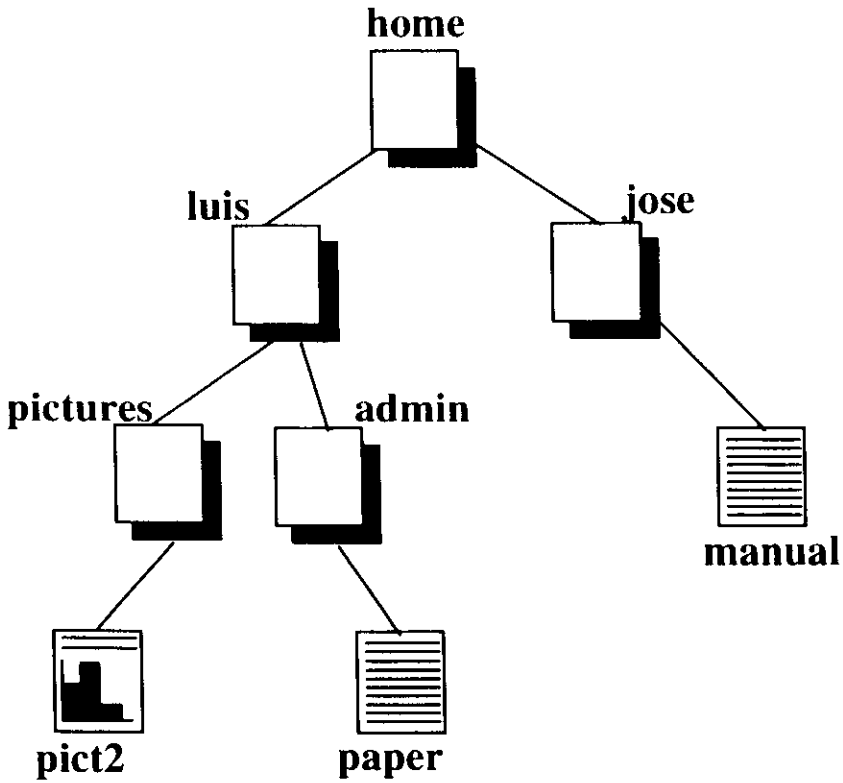
more paper

cat paper

tail paper

mv paper admin/

wc paper



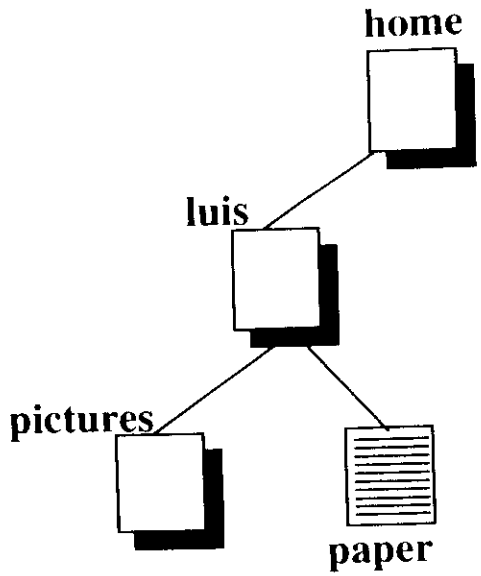
Working with directories

- **Moving around**

  pwd
  mkdir pictures
  cd pictures
  cd ..
  rmdir pictures

- **Finding files**

  find / -name .login -print



Communicating with others

talk jose@nostradamus

write luis

wall

mail

Useful commands

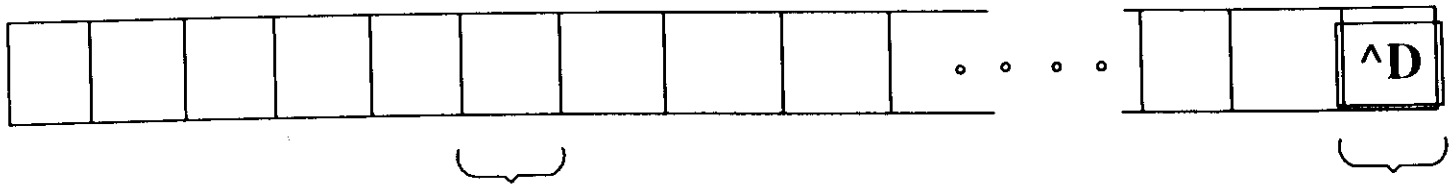
cal date who

man ping

finger luis

ping nostradamus

The file concept

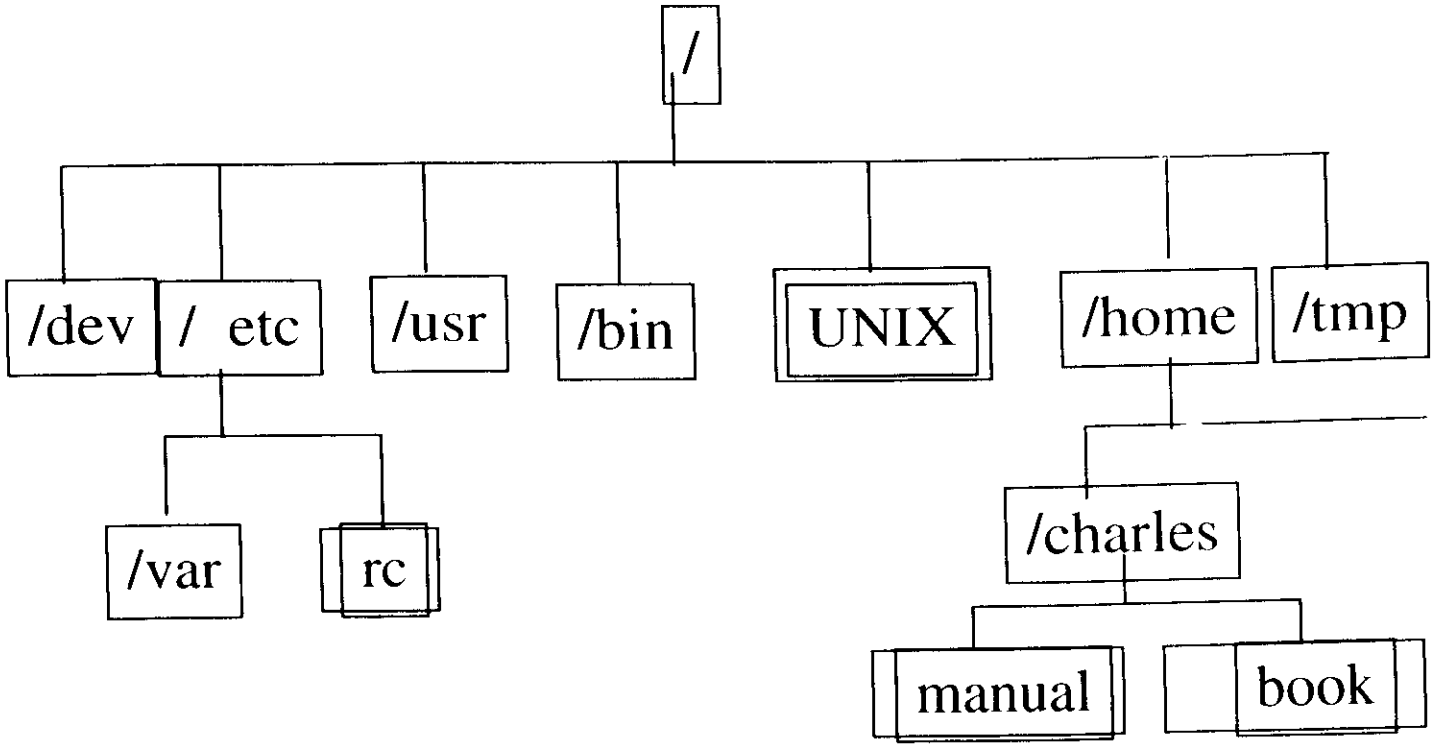


byte

u octet

Control D

The file system



The root directory and its branches

/ → unix

/bin

/dev

/etc → passwd,inetd.conf,host

/lib → uucp, cron

/usr

/usr/man

/home

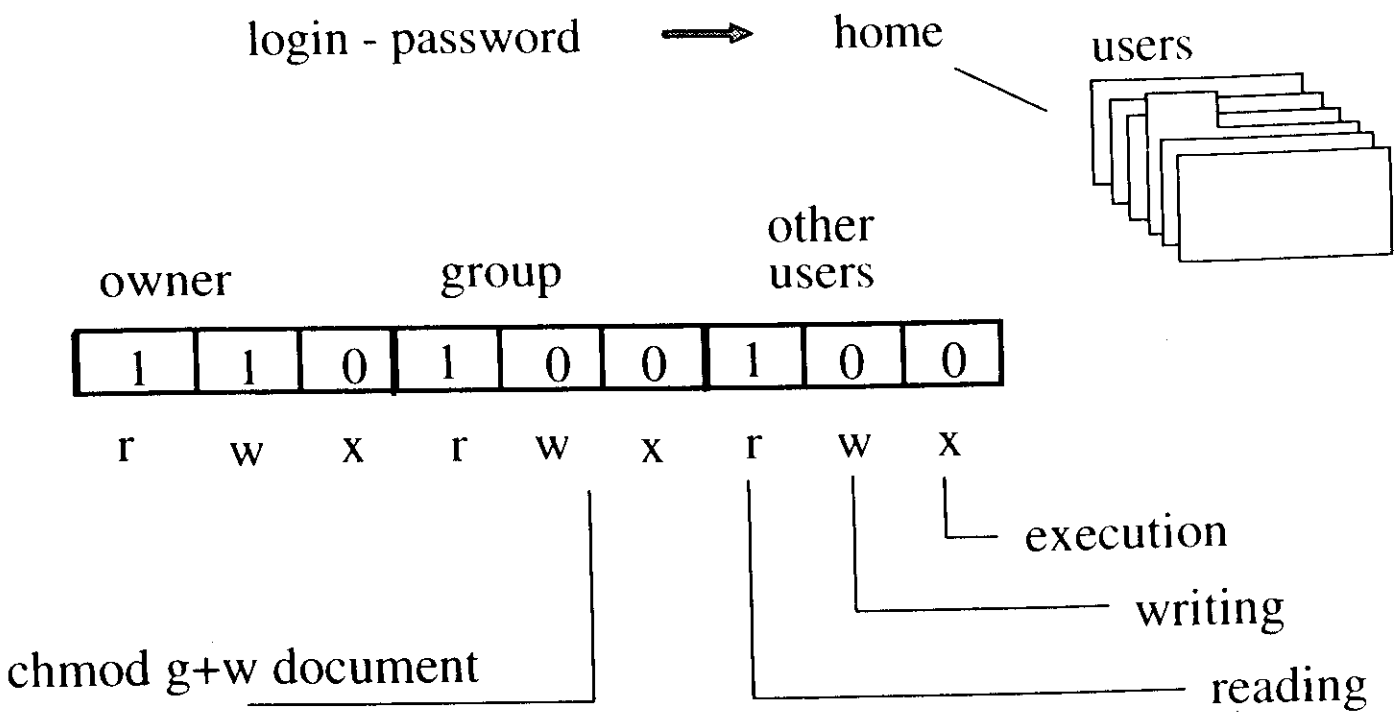
/usr/spool

/usr/spool/lp

/usr/spool/mail

/usr/spool/uucp

File permissions



File ownership

chown user1 file

Default permissions

umask 011

Protecting your information

- File encryption

  crypt

- Keep the key secret

- File unencryption

Security guidelines

- Choose an appropriate password and protect it
- Encrypt any confidential files
- Use the right permissions
- Protect your configuration files
- Log out properly
- Never leave an open session unattended

The VI editor

The working area

In protohistoric times, the ancient town of Teergeste was very probably the cite of a Casttelliere, where people of Venetian, Carnic, Gallic and Celtic origin would meet.

~

~

~

~

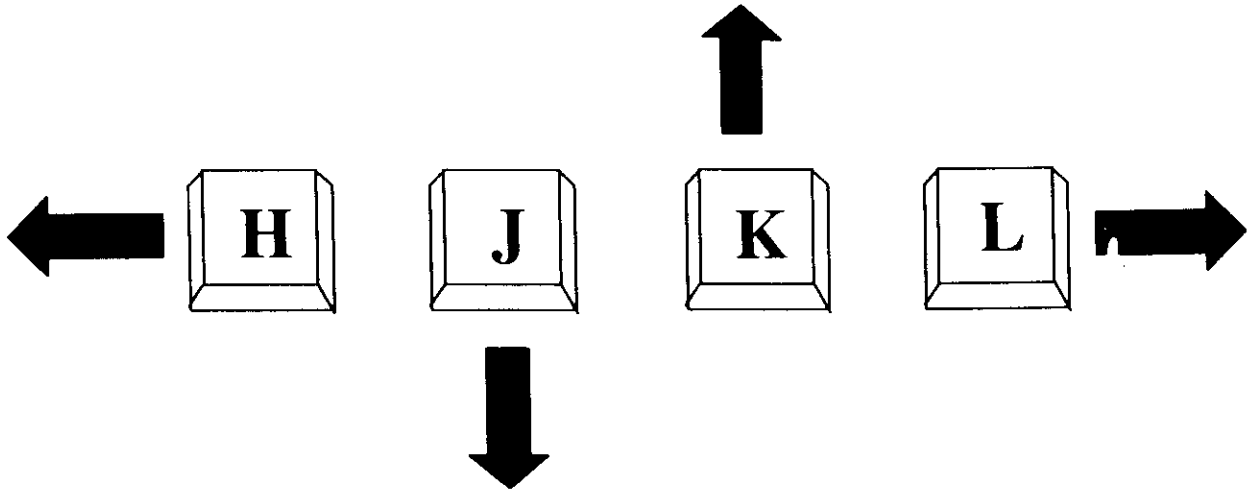
~

: w

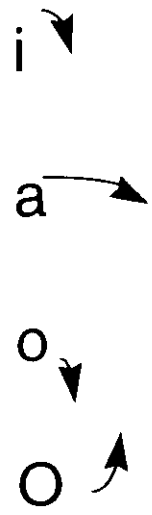
Adjusting the terminal type

```
$vi paper1
Ansi:Unknown terminal type
I don't know what kind of terminal you are  on
[Using open mode]
"paper1" [Read only] 286 lines, 14247 characters
```

Moving around



Switching to insert mode



In protohistoric times, the ancient town of Teergeste was very probably the city of a Casttelliere, where people of Venetian, Carnic, Gallic and Celtic origin would meet.

~

~

~

~

~

Deleting text

dw

dd

d0

d$

In protohistoric times, the ancient town of Teergeste was very probably the city of a Casttelliere, where people of Venetian, Carnic, Gallic and Celtic origin would meet.

~

~

~

~

~

Moving Text

Y

yw

p

P

In protohistoric times, the ancient town of Teergeste people of Venetian, Carnic, Gallic and Celtic origin would meet.

was very probably the city of a Casttelliere, where

~

~

~

~

~

Saving your work

save and continue ~ ~ : w	save and exit ~ ~ : wq
save even if read only ~ ~ : w!	quit without saving ~ ~ : q!

Search and replace

/pattern ?pattern

4,$s/old word/new text/g

Common VI commands

vi *file*

vi -r *file*

h j k l

ENTER

0

$

^U ^D

^B ^F

^L

a ESC

A ESC

i ESC

I ESC

o ESC

O ESC

b

w

e

dw

cw ESC

dd

cc ESC

D

C ESC

x

s ESC

S ESC

r

R ESC

/

?

n

N

u

U

y

Y

p

P

:w

:wq

:q

:q!

: *number*

ZZ

:set *mode*

Advanced UNIX commands

Working with files

Unix is a multiuser, multitasking operation system that enables different people to access a computer at the same time.

```
cut -c10-20 /etc/file
```

multiuser,
system the
people to
t the same

Working with files

book	jhon
car	matthew
file	frank
computer	carol
tv	kim
chair	time

paste file-a file-b

book	jhon
car	matthew
file	frank
computer	carol
tv	kim
chair	time

Finding differences

Finding the differences

Finding THE differences

diff file-a file-b

```
1c1
< Finding the differences
--
> Finding THE differences
```

Sorting files

Unix is a multiuser, multitasking
operation system that enables
different people to access a
computer at the same time.

sort /etc/file

Unix is a multiuser, multitasking
computer at the same time.
different people to access a
operation system that enables

Finding duplicates

apples
apples
banana
kiwi
kiwi
watermelon

uniq /etc/file

apples
banana
kiwi
watermelon

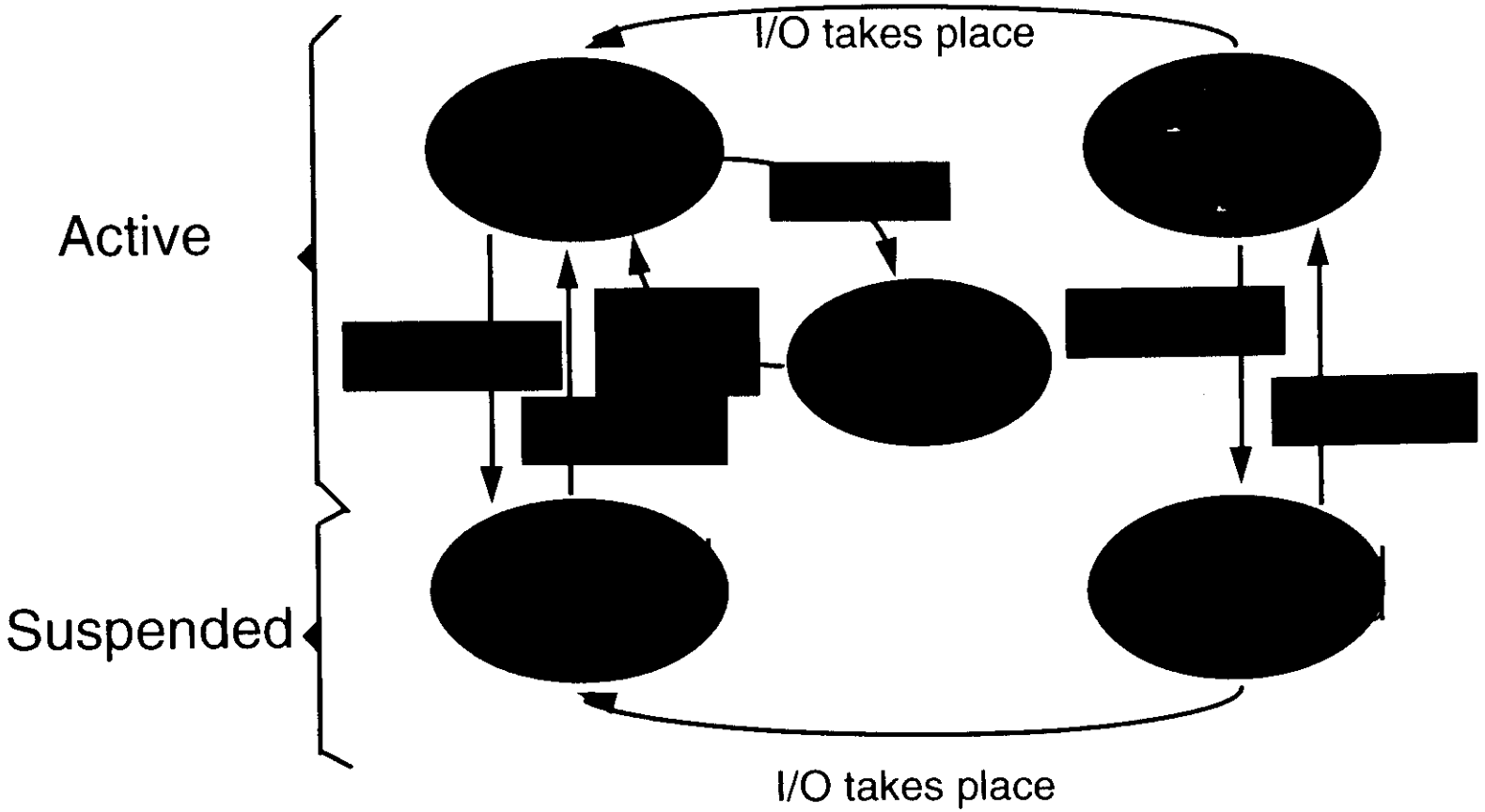
Searching information

Unix is a multiuser, multitasking
operation system that enables
different people to access a
computer at the same time.

grep at /etc/file

operation system that enables
computer at the same time.

Managing processes



Controlling Jobs

Ctrl-Z

bg

fg

jobs

kill

stop

&

Advanced concepts

- Several commands on the same line

  command ; command

- Pipelining

  command1 | command2 | command3

Metacharacters

>	;	\
>>	&	' '
<	` `	" "
|	( )	#
*	{ }	*var = valor*
?	$1,...	*proc1 && proc2*
[*ccc*]	$*var*	*proc1 || proc2*
	${*var*}	

Creating new commands

- Several commands can be executed sequentially by w·iting them in a file and giving it permission to execute.

- The shells allow you to use a very powerful programming language.

Shell programming language

Variables

```
city = Mérida
message = "meeting at 3 p.m."

export city
echo $city
```

Shell programming language

```
names = `cat list`
echo Today's date is `date`
echo The number of arguments is$#
chmod +x $1
```

Making decisions

if *expression*
then

    Commands to execute if expression is true

else

    Commands to execute if expression is false

fi

Loops

"for"

```
for variable in list
do
        commands
done
```

"while"

```
while command
do
        commands
done
```

"until"

```
until command
do
        commands
done
```

A shell program

```
#
for name in `cat personal`
do
        echo Hello $name
done
```

Unix Tutorial

Contents:

Logging On Remotely (e.g. from a sun)

Hotrod is available for login from the Sun Lab on the fifth floor. Log into the sun and then from an Xterm window type:

```
rlogin hotrod
```

You can then follow the instructions for logging on below.

Logging On

When you first connect to one of the Unix computers you will see the prompt:

```
login:
```

If you see only the prompt `Password:` you probably used `rlogin`. `rlogin` assumes that your username is the same on all computers and enters it for you. If your username is different, don't worry, just press <CR> until you see the `login:` prompt and start from scratch.

At the `login:` prompt, type in your username. *Be careful to type only lower case!* The Unix operating system is "case sensitive." If you type your username in mixed case (`Rarmour` rather than `rarmour`, for example) the computer will not recognize it.

Your Password

Once you have typed in your username you will be prompted to type in your password. Type carefully! It won't be displayed on the screen.

The motd

If you type your username and password correctly, the computer will begin running the login program. It starts by displaying a special message of the day contained in the `/etc/motd` file. This file will usually contain information about the computer you are logging onto, maybe a basic message about getting help, and any important system messages from the system manager.

Initialization Files

When you log in, the Unix login program starts up a command shell. Users do not deal with the operating system directly. Instead they interact with a shell, which is initialized with several pieces of information such as your username, home directory and "path." By default all users use the tcsh-shell (the program `/bin/tcsh`) and interact with it.

There are a couple of files read by this shell when your login session starts up. These are the `.cshrc` file and the `.login` file. These files are created when your account is created. As you learn more about how Unix and the tcsh-shell work, you may want to customize these files.

Using the System

Finally you are logged in! You will see a prompt like the following:

```
>ell notrod  matmod> >or >hotrod %
```

just waiting for you to type something. Throughout the Unix Tutorial guide we will use `%` to indicate the computer's ''ready'' prompt.

ls

Okay, let's try a simple command. Type `ls` and press <CR>. `ls` lists files in a directory. Right now you may or may not see any files, but not seeing any files doesn't mean that you don't have any! `ls` by itself won't list hidden files (files whose names start with ''.'', like `.login`). Now try typing:

```
% ls -a
```

Don't actually type the `%` symbol! Remember, that's the computer's prompt which indicates it is ready to accept input. The spacing should be exactly as shown: `ls` followed by a space, followed by `-a`. The `-a` is a ''flag'' which tells the `ls` program to list all files.

For more about command flags see below.

cd

Just for fun, let's look at the contents of another directory, one with lots of files. Directory names in Unix are straightforward. They are all arranged in a tree structure from the root directory ''/''.

For now, use `cd` to change your directory to the `/bin` directory. Type:

```
% cd /bin
```

and press <CR>. Now type `ls` again. You should see a long list of files. In fact, if you look carefully you will see files with the names of the commands we've been typing (like `ls` and `cd`). Note that the `/bin` in the command we typed above was not a flag to `cd`. It was a ''parameter.'' Flags tell commands how to act; parameters tell them what to act on.

Now return to your home directory with:

```
% cd
```

Entering `cd` with no parameter returns you to your home directory. You can check to make sure that it worked by entering:

```
% pwd
```

which prints your current (or ''working'') directory. The computer will return a line of words separated

by ''/'' symbols which should look something like:

`home` *username*

Whatever it returns, the list should end in your username.

Using the On-Line Manual Pages

Most Unix commands have very short and sometimes cryptic names like `ls`. This can make remembering them difficult. Fortunately there are on-line manual pages which allow you to display information on a specific program (to list all the flags of `ls`, for example) or list all the information available on a certain topic.

man

To investigate other flags to the `ls` command (such as which flags will display file size and ownership) you would type `man ls`.

man -k

The second way of using the on-line manual pages is with `man -k`. In this case you use a word you expect to be in a one-line description of the command you wish to find. To find a program which ''lists directory contents'' you might type `man -k dir`. Partial words can be used and this is one of the few places in Unix where upper and lower case are allowed to match each other.

Using man and more

Try it now. Use `man ls` to find out how to make the `ls` program print the sizes of your files as well as their names. After typing `man ls` and pressing <CR>, note how `man` displays a screenful of text and then waits with a prompt `--More--` at the bottom of the screen.

What `man` is doing is sending everything it wants to display on the screen through a program known as a ''pager.'' The pager program is called `more`. When you see `--More--` (in inverse video) at the bottom of the screen, just press the space bar to see the next screenful. Press <CR> to scroll a line at a time.

Have you found the flag yet? The `-s` flag should display the size in kilobytes. You don't need to continue paging once you have found the information you need. Press `q` and `more` will exit.

Listing File Sizes

Now type `ls -as`. You can stack flags together like this. The command `ls -as` lists all files, and lists their sizes in kilobytes.

Logging Off

When you are finished you should be sure to log out! You need to be careful that you've typed `logout`

correctly. The Unix operating system is not forgiving of mistyped commands. Mistyping `logout` as ''logotu'', pressing return and then leaving without glancing at the screen can leave your files at anyone's mercy.

The Unix Shell Syntax

As mentioned earlier, user commands are parsed by the shell. There are many shells other than the the C-shell which allow different types of shortcuts. We will only discuss the C-shell here, but some alternate shells installed on SG's include the Bourne shell (`/bin/sh`), the and the `tcsh` (a C-shell variant). While you are welcome to experiment with any of these shells, realize that tcsh-shell is the default shell and that some utilities assume you are using it (and may not work if you're not).

The Path

One of the most important elements of the shell is the path. Whenever you type something at the `%` prompt, the C-shell first checks to see if this is an alias you have defined and, if not, searches all the directories in your path to determine the program to run.

The path is just a list of directories, delimited by colons, which are searched in order. Your default `.cshrc` will have a path defined for you. If you want other directories (such as a directory of your own programs) to be searched for commands, add them to your path by editing your `.cshrc` file. This list of directories is stored in the PATH environment variable. We will discuss how to manipulate enviroment variables later.

Flags and Parameters

Most commands expect or allow parameters (usually files or directories for the command to operate on) and many provide option flags. A flag, as we saw before, is a character or string with a - before it such as the `-s` we used with the `ls` command.

Some commands, such as `cp` and `mv` require file parameters. Not surprisingly, `cp` and `mv` (the copy and move commands) each require two: one for the original file and one for the new file or location.

It would seem logical that if `ls` by itself just lists the current directory then `cp` *filename* should copy a file to the current directory. Instead you must enter `cp` *filename* . where the ''.'' tells `cp` to place the file in the current directory. *filename* in this case would be a long filename with a complete directory specification.

Not surprisingly, `ls .` and `ls` are almost the same.

Directory and File Structure

Directories

Directories in Unix start at the root directory "/". Files are fully specified when you list each directory branch needed to get to them.

```
/usr/local/lib/news

/home/pamela/src/file.c
```

Your Home Directory

A home directory can always be specified with *~username* If you needed to list files in someone else's home directory, you could do so by issuing the command:

```
% ls ~username
```

substituting in their username. You can do the same with your own directory if you've `cd`'d elsewhere. Please note: many people consider looking at their files an invasion of privacy, even if the files are not protected. Just as some people leave their doors unlocked but do not expect random passers-by to walk in, other people leave their files unprotected without intending to invite browsers.

Subdirectories

If you have many files or multiple things to work on, you probably want to create subdirectories in your home directory. This allows you to place files which belong together in one distinct place.

Creating Subdirectories

The program to make a subdirectory is `mkdir`. If you are in your home directory and wish to create a directory, type the command:

```
% mkdir directory-name
```

Once this directory has been created you can copy or move files to it (with the `cp` or `mv` commands) or you can `cd` to the directory and start creating files there.

Copy a file from the current directory into the new subdirectory by typing:

`cp` *filename directory-name*
: copy file, filename will be the same as original

`cp` *filename directory-name/new-filename*
: copy file, give it a new name

Or `cd` into the new directory and move the file from elsewhere:

```
% cd directory-name
```

```
% cp ../filename .
```

copies the file from the directory above (represented by "..") to the current directory (represented by

``.''), giving it the same filename.

File Names

Unlike other operating systems, filenames are not broken into a name part and a type part. Names can be many characters long and can contain most characters. Some characters such as * and ! have special meaning to the shell. They should not be used in filenames. If you ever do need to use such a symbol from the shell, they must be specified sneakily, by ''escaping'' them with a backslash (\). For example:

```
% rm  !badfile
```

C-shell would have interpreted `rm !badfile` differently. In that case, `!badfile` would have been replaced with the last command beginning with ''badfile.'' Chances are no such command would have existed, resulting in the error message `badfile: Event not found`. See the section on history for more information.

Specifying Files

There are two ways to specify files:

- fully, in which case the name of the file includes all of the directories, starting from the root director, ''/'', or
- relatively, in which case the filename starts with the name of a subdirectory or consists solely of its own name.

When Charlotte Lennox (username `lennox`) created her directory `arabella`, all of the following sets of commands could be used to display the same file:

```
% more ~lennox/arabella/chapter1
```

or

```
% cd ~lennox
% more arabella chapter1
```

or

```
% cd ~lennox/arabella
% more chapter1
```

The full file specification, beginning with a ''/'' is very system dependent. On the CCO Unix Cluster, a user directories are ''automounted'' on the `/home` partition. This means that `~lennox`on the CCO Unix Cluster would be the same as `/home/lennox` and that `chapter1` would be fully specified by:

```
/home/lennox/arabella/chapter1
```

Disk Space Maintenance

It's important to keep track of how much disk space you are using. The command `du` displays the disk usage of the current directory and all of its subdirectories. It displays the usage, in kilobytes, for each directory, including any subdirectories it contains, and ends by displaying the total.

`% du`
: Display disk usage of the current directory and its subdirectories.

`% du -s`
: Display only total disk usage.

`% du -s -k`
: Some versions of Unix, such as Solaris, need -k to report kilobytes. Currently, the machines `stucco` and `raccoon` run Solaris.

The df Program

To examine what disks and partitions exist and are mounted, you can type the `df` command at the `%` prompt. This should display partitions which have names like `/dev/sd3g`: `3` for disk 3, `g` for partition g. It will also display the space used and available in kilobytes and the "mount point" or directory of the partition.

Scratch Space

Users have home directories for storing permanent files. At various busy times of the year there may be shortages of disk space on the Unix Cluster. You should use the `du` command to stay aware of how much space you are using and not exceed the system limits. Currently, users are expected not to use more than 5 megabytes of disk space.

Temporary scratch space is available in the event of a disk crunch, however, and can be used to store files which are extremely large or relatively unimportant. The scratch space is located in the `/ccovol.suntmp` and `/ccovol.nexttmp` directories. You should use the `mkdir` program to create a directory for yourself on either of these partitions.

Remember, because the scratch space is for temporary storage, you should delete the files as soon as you are done with them. This is *particularly* important if your files are large. If you forget to remove your files, they will be removed for you, but not until seven or more days after you last access them.

Displaying owner, group and permissions

The command `ls -lg` *filename* will list the long directory list entry (which includes owner and permission bits) and the group of a file.

The display looks something like:

```
permission  owner       group       filename
-rw-r-----  hamilton    ug          munster_village
```

Protecting Files and Directories

When created, all files have an owner and group associated with them. The owner is the same as the username of the person who created the files and the group is the name of the creator's default login group, such as `faculty`, `grads`, `ug`, etc.

Most users belong to one group on the CCO Unix computers, such as `ug` or `faculty`. If the owner of the file belongs to more than one group (you can display the groups to which you belong with the `groups` command) then the owner can change the group of the file between these groups. Otherwise, only the `root` account can change the group of a file.

Only the `root` account can change the ownership of a file.

The Permission Bits

The first position (which is not set) specifies what type of file this is. If it were set, it would probably be a `d` (for directory) or `l` (for link). The next nine positions are divided into three sets of binary numbers and determine permissions for three different sets of people.

```
u       g       o
421     421     421
rw-     r--     ---
 6       4       0
```

The file has "mode" 640. The first bits, set to "`r` + `w`" (4+2=6) in our example, specify the permissions for the user who owns the files (u). The user who owns the file can read or write (which includes delete) the file.

The next trio of bits, set to "`r`" (4) in our example, specify access to the file for other users in the same group (g) as the group of the file. In this case the group is `ug` -- all members of the `ug` group can read the file (print it out, copy it, or display it using `more`).

Finally, all other users (o) are given no access to the file.

The one form of access which no one is given, even the owner, is "`x`" (for execute). This is because the file is not a program to be executed. It is probably a text file which would have no meaning to the computer. The `x` would appear in the third position.

Changing the Group and the Permission Bits

The group of a file can be changed with the `chgrp` command. Again, you can only change the group of a file to a group to which you belong. You would type as follows:

```
% chgrp groupname filename
```

You can change the protection mode of a file with the `chmod` command. This can be done relatively or absolutely. The file in the example above had the mode 640. If you wanted to make the file readable to all other users, you could type:

```
% chmod 644 filename
```

or

```
% chmod o+r filename
```

or

```
% chmod +4 filename
```

For more information see the man page for chmod.

Default Permissions: Setting the umask

All files are assigned a default set of permissions. To set the default, you must set the value of the variable umask. umask must be defined once per login (usually in the .cshrc file). Common umask values include 022, giving read and execute (or directory search) but not write permission to the group and others and 077 giving no access to group or other users for all new files you create. Note that the umask bits represent permissions *not* to be given (i.e. the opposite of what ls -l would show).

Creating Files

The cat Program

cat is one of most versatile commands. The simplest use of cat:

```
% cat .cshrc
```

displays your .cshrc file to the screen. Unix allows you to redirect output which would otherwise go to the screen by using a > and a filename. You could copy your .cshrc, for example, by typing:

```
% cat .cshrc > temp
```

This would have the same effect as:

```
% cp .cshrc temp
```

More usefully, cat will append multiple files together.

```
% cat .cshrc .login > temp
```

will place copies of your .cshrc and .login into the same file. Warning! Be careful not to cat a file onto an existing file! The command:

```
% cat .cshrc > .cshrc
```

may *destroy* the file .cshrc.

If you fail to give `cat` a filename to operate on, cat expects you to type in a file from the keyboard. You must end this with a <Ctrl>-D on a line by itself. <Ctrl>-D is the end-of-file character.

By combining the above two concepts, leaving off the name of a file to input to `cat` and telling `cat` to direct its output to a file with `>` *filename*, you can create files. For example:

```
% cat > temp

;klajs;dfkjaskj
alskdj;kjdfskjdf
<Ctrl>-D
%
```

This will create a new file `temp`, containing the lines of garbage shown above. Note that this creates a new file. If you want to add things on to the end of an existing file you must use `cat` slightly differently. Instead of `>` you'd use `>>` which tells the shell to append any output to an already existing file. If you wanted to add a line onto your `.cshrc`, you could type

```
% cat >> .cshrc
echo "blah blah blah"
<Ctrl>-D
%
```

This would append the line `echo "blah blah blah"` onto your `.cshrc`. Using `>` here would be a bad idea; it might obliterate your original `.cshrc` file.

Files as Output and Log Files

Ordinarily there are two types of output from commands: output to standard output (stdout) and to standard error (stderr). The `>` and `>>` examples above directed only standard output from programs into files. To send both the standard output and error to a file when using the C-shell, you should type `>&`:

```
% command >& filename
```

Logging Your Actions to a File

Sometimes you may wish to log the output of a login session to a file so that you can show it to somebody or print it out. You can do this with the `script` command. When you wish to end the session logging, type `exit`.

When you start up you should see a message saying `script started, file is typescript` and when you finish the script, you should see the message `script done`. You may want to edit the typescript file: visible ^M's get placed at the end of each line because linebreaks require two control sequences for a terminal screen but only one in a file.

Text Editors

`cat` is fine for files which are small and never need to have real changes made to them, but a full-fledged editor is necessary for typing in papers, programs and mail messages. Among the editors

available on the CCO Unix computers are `pico`, `vi` and `emacs`.

Be careful: not all Unix editors keep backup copies of files when you edit them.

pico

`pico` is a simple, friendly editor. Type `setup pine` to set it up, `pico` *filename* to start it and `man pine` for more information about how to use it.

vi

`vi` is an editor which has a command mode and a typing mode. When you first startup `vi` (with the command `vi` *filename*) it expects you to enter commands. If you actually want to enter text into your file, you must type the insert command `i`. When you need to switch back to command mode, hit the escape key, usually in the upper left corner of your keyboard.

To move around you must be in command mode. You can use the arrow keys or use `j, k, h, l` to move down, up, left and right.

For more information type `man vi`. There is a reference sheet containing lists of the many `vi` commands across from the CCO Front Office in Jorgensen.

Emacs

Emacs is a large editing system. Copies of the two-page reference sheet are available in the reference sheet rack across from the Front Office.

To use `emacs`, type:

```
% setup emacs
% emacs
```

Searching Through Files

The `grep` program can be used to search a file for lines containing a certain string:

`% grep` *string filename*

`% grep -i` *string filename* (case insensitive match)

or not containing a certain string:

`% grep -v` *string filename*

See the `man` page for `grep`. It has many useful options.

`more` and the `vi` editor can also find strings in files. The command is the same in both: type */string* when at the `--More--` prompt or in `vi` command mode. This will scroll through the file so that the line with

"string" in it is placed at the top of the screen in `more` or move the cursor to the string desired in `vi`. Although `vi` is a text editor there is a version of `vi` called `view`, which lets you read through files but does not allow you to change them.

Comparing Files

The basic commands for comparing files are:

`cmp`
: states whether or not the files are the same

`diff`
: lists line-by-line differences

`comm`
: three column output displays lines in file 1 only, file 2 only, and both files

See the `man` pages on these for more information.

File Types

When you list files in Unix, it can be very hard to tell what kind of files they are. The default behavior of the `ls` program is to list the names of all the files in the current directory without giving any additional information about whether they are text files, executable files or directories. This is because the meaning of the contents of each file is imposed on it by how you use the file. To the operating system a file is just a collection of bytes.

There is a program `file` which will tell you information about a file (such as whether it contains binary data) and make a good guess about what created the file and what kind of file it is.

The System and Dealing with Multiple Users

Most Unix commands which return information about how much CPU time you've used and how long you've been logged in use the following meanings for the words "job" and "process."

When you log in, you start an interactive "job" which lasts until you end it with the `logout` command. Using a shell like C shell which has "job control" you can actually start jobs in addition to your login job. But for the purposes of the most information returning programs, "job" refers to your login session.

Processes, on the other hand, are much shorter lived. Almost every time you type a command a new process is started. These processes stay "attached" to your terminal displaying output to the screen and, in some cases (interactive programs like text editors and mailers), accepting input from your keyboard.

Some processes last a very long time -- for example, the `/bin/csh` (C-shell) process, which gets started when you log in, lasts until you log out.

Information About Your Processes

You can get information about your processes by typing the `ps` command.

```
  PID TT STAT   TIME COMMAND
 9980 s9 S      0:06 -csh (csh)
12380 s9 R      0:01 ps
```

The processes executing above are C-shell and the `ps` command. Note that both commands are attached to the same terminal (TT), have different process identification numbers (PID), and have different amounts of CPU time (TIME), accumulated.

Information About Other People's Processes

who

The simplest and quickest information you can get about other people is a list of which users are logged in and at which "terminals" (terminal here is either a terminal device line or `telnet` or `rlogin` session). The command to do this is `who` and it responds quickest of all the commands discussed here because it simply examines a file which gets updated every time someone logs in or out.

Be careful though! This file, `/etc/utmp`, can get out of date if someone's processes die unexpectedly on the system. Any program which uses `utmp` to report information might occasionally list users who are not really logged in!

```
% who
johnjohn ttyp2    Aug 25 15:31   (unholy1.caltech.)
mikejcai ttyp4    Aug 26 05:55   (RuddockNIU-246.c)
mikejcai ttyp7    Aug 26 06:30   (RuddockNIU-246.c)
billgr   ttyp9    Aug 26 09:17   (argon.caltech.ed)
bob      ttypa    Aug 22 09:38   (logan.caltech.ed)
tlynch   ttypb    Aug 26 09:17   (romeo.caltech.ed)
vonsrdmn ttypf    Aug 24 21:04   (Sun.COM)
marcel   ttyq2    Aug 26 06:09   (montana.caltech.)
johnjohn ttyq3    Aug 25 15:49   (unholy1.caltech.)
ashaw    ttyq4    Aug 26 08:34   (131.215.4.23)
heather  ttyq5    Aug 24 15:56   (thistle)
tnt      ttyq7    Aug 26 09:24   (onion-rings.calt)
gchoi    ttyq8    Aug 26 08:26   (131.215.212.5)
rajan    ttyqc    Aug 26 08:40   (gw1.octel.com)
wassgren ttyqd    Aug 26 09:31   (131.215.174.17)
albwang  ttyqe    Aug 26 09:11   (131.215.4.112)
trevor   ttyqf    Aug 26 08:31   (sense.caltech.ed)
jzl      ttyr0    Aug 26 09:27   (wolfgang.caltech)
%
```

w

The `w` command is slower than the `who` command because it returns more information such as details about what programs people are running. It also returns a line containing the number of users and the system load average. The load average is the average number of processes ready to be run by the CPU and is a rough way of estimating how busy a system is.

```
% w
  9:17am  up 4 days, 4 mins,  18 users,  load average: 5.81, 5.73, 5.52
User     tty      login@  idle   JCPU   PCPU  what
jonnjohn ttyp2    3:31pm     5     49     31  elm
mikejcai ttyp4    5:55am          1:36      4  ftp ifcss.org
mikejcai ttyp7    6:30am     6    1:37      5  -tcsh
billgr   ttyp9    9:17am          12        9  trn
bob      ttypa   Mon 9am    45    7:57     11  -csh
tlyncn   ttypb    9:17am          17       11  nn
vonsrdmn ttypf   Wed 9pm      7    1:09     13  -tcsh
marcel   ttyq2    6:09am      1      19      9  nn
jonnjohn ttyq3    3:49pm     18    1:14      1  -csh
asnaw    ttyq4    8:39am     43       3      1  -csh
heather  ttyq5   Wed 3pm     33    4:39      1  -csh
tnt      ttyq7    9:24am          16        3  elm
gchoi    ttyq8    8:26am      6     31       9  elm
rajan    ttyqc    8:40am     30     17       7  nn
jjmach   ttyqd    9:10am      1     30      15  nn
albwang  ttyqe    9:11am           5       2  -tcsh
trevor   ttyqf    8:31am          20      19  nn
jzl      ttyr0    9:27am           3       1  pine
%
```

ps

The `ps` command used earlier to list your own processes can be used to list other users' processes as well. `who` and `w` list logins but not individual processes on the system. They don't list any of the running operating system processes which start when the computer is booted and which don't have logins.

Since `ps` doesn't use `utmp`, it is the program to use when you really want to find out what processes you might have accidentally left on the system or if another user is running any processes. Note that although `ps` might report processes for a user, it might be because that user has left a "background job" executing; the user is not really logged in. In this case you should see a "?" in the TT field.

To get this fuller listing, use `ps -aux` under SunOS or `ps -ef` under Solaris. For more information on the uses of `ps`, type `man ps`.

finger

The `finger` program returns information about other users on the system who may or may not be logged in. `finger` by itself returns yet another variation of the list of currently logged in users. `finger` followed by a username or an e-mail -style address will return information about one or more users, the last time they logged into the system where you are fingering them, their full name, whether or not they have unread mail and the contents of two files they may have created: `.plan` and `.project`

For more information about using `finger` or ways to provide information about yourself to others, type `man finger`.

```
% finger
Login       Name                    TTY Idle    When    Where
jonnjohn John Carlos White           p2    5 Thu 15:31  unholy1.caltech.
mikejcai Jun Cai                     p4      Fri 05:55  RuddockNIU-246.c
mikejcai Jun Cai                     p7    5 Fri 06:30  RuddockNIU-246.c
```

```
xxxxxx   Joseph D. Billock      p9       Fri 13:17   argon.caltech.ed
bob      Robert S. Logan        pa    45 Mon 19:35   logan.caltech.ed
tlynch   Timothy W. Lynch       pb       Fri 09:17   romeo.caltech.ed
vonsrdmn Srdjan Sobajic         pf     7 Wed 21:04   Sun.COM
marcel   Marcel Bergmann        q2       Fri 06:09   montana.caltech.
johnjohn John Carlos White      q3    17 Thu 15:49   unholy1.caltech.
ashaw    Amy Shaw               q4    42 Fri 08:39   131.215.4.23
heather  Heather L. Sherman     q5    32 Wed 15:58   thistle
tnt      Thann-nga Tran         q7       Fri 09:24   onion-rings.calt
gchoi    Garrett Choi           q8     8 Fri 08:26   131.215.212.5
rajan    Rajan Ranga            qc    30 Fri 08:40   gw1.octel.com
jjmach   Jeffrey Mach           qd       Fri 09:10   hyper.galcit.cal
albwang  Chun-Ming Wang         qe       Fri 09:11   131.215.4.112
trevor   Trevor Roper           qf       Fri 08:31   sense.caltech.ed
jzl      James Z. Lee           r0       Fri 09:27   wolfgang.caltech
%
```

Sending Messages and Files to Other Users

Electronic mail programs run on almost all the computers at Caltech and usually have two parts: a user interface which lets users read and send messages and a system mailer which talks to mailers on other computers. This mailer receives outgoing messages from the user interface programs and delivers incoming messages to the user mailbox (which the interface program reads).

/usr/ucb/mail

There are many user interfaces available on the Unix computers, all of which provide similar functionality. The program supplied with most Unix computers is `/usr/ucb/mail` (or `Mail`). To read messages type `Mail`, to send messages type:

`% Mail` *address*

For information about valid mail addresses, please see the Electronic Mail Guide. Note that on Solaris machines the command `Mail` has been changed to `mailx`.

You should next see a `Subject:` prompt. If you don't see a prompt, don't worry, just type in your one line subject anyway and press return. You may start typing your message (but you will be unable to correct errors on lines after you have pressed <CR> to move to the next line) or you may may specify a file to include with *~r filename*.

You may invoke a text editor like `vi` by typing *~v* If you wish regularly to use an editor other than `vi` you should see the information later in the section about enviroment variables.

There are many other commands you may enter at this point -- see the `Mail` man page for all of them. When you are finished typing in your message (if you have used *~v* to run a text editor, you should exit from it) press <Ctrl>-D on a line by itself. Most likely you will now see a `Cc:` prompt. If you wish to send copies of your message to someone besides the recipient you would enter the address or addresses (separated by commas) and press return. Otherwise press return without entering an address.

Other Mail Programs

Pine is a full-screen interactive mailer that is very straightforward to use. It is currently installed on the CCO Unix Cluster Suns. If you have the command `setup pine` in your `.cshrc` then you should type `pine`. If the `pine` command fails you probably need to type `setup pine` first. For more information type `man pine`.

Other mailers include ELM and MH. Both require `setup` commands. The command to start ELM is `elm`. MH is acutally a set of several separate programs. See `man mh` for more information. Documents about using Pine, ELM and MH are available from CCO. To get a copy, send mail to `root`.

Write

The `write` program can be used to send messages to other users logged onto the system. It's not the best way of having a conversation, but it's simple to use. Enter:

```
% write username
```

and you can start writing lines to the terminal of the person you want to send messages to. The person must be logged in, and, if they are logged in more than once, you must specify the terminal to `write` to; `write melville ttyh1`, for example.

Talk

`talk` is a program which allows two users to hold a conversation. Unlike `write`, it can be used between different computers; and, unlike `write`, it divides the screen so that the things you type appear in the top half and the things written to you appear in the bottom half. There is also a program called `ytalk` which we recommend that people use instead of the standard `talk`.

To `talk` to users on the same computer:

```
% talk username
```

To `talk` to users on another computer use the address format of *username@nodename*:

```
% talk brunton@jarthur.claremont.edu
```

Addressing Remote Nodes

`talk` can only be used to other Internet nodes: usually computers which have ending names such as .edu, .com, .org, .gov, or .mil. Not all computers with these names are attached directly to the Internet. `finger` and `talk` won't work with computers which are only attached by mail gateways. For more information on the Internet, see the Campus Network Guide.

Shortcuts

If you use certain command flags regularly (`-lga` for `ls`) you can `alias` them to shorter commands.

You can use wildcard symbols to refer to files with very long names. You can easily repeat commands you have already executed or modify them slightly and re-execute them.

Aliases

As mentioned above, you can `alias` longer commands to shorter strings. For example, `ls -F` will list all the files in the current directory followed by a trailing symbol which indicates if they are executable commands (*) or directories (/). If you wanted this to be the default behavior of `ls` you could add the following command to your `.cshrc`:

```
% alias ls ls -F
```

To list the aliases which are set for your current process, type:

```
% alias
```

without any parameters.

Wildcards

Wildcards are special symbols which allow you to specify matches to letters or letter sequences as part of a filename.

Some examples:

*

matches zero or more characters

`ls *.dat`
lists all files ending in `.dat`

`ls r*`
lists all files starting with `r`

Beware of the `rm *` command!

?

matches one character

`ls ?.dat`
lists `5.dat`, `a.dat`, but not `70.dat`

[]

matches one of the characters inside the brackets

`ls *.[ch]`
lists all `.h` and `.c` files

`more [Rr][Ee][Aa][Dd][Mm][Ee]`
mores the files `README`, `readme`, `ReadMe` and `Readme`, among others

Directory Specifications

You've already met the ~ shortcut. The two other important directory symbols are "." for the current

directory and ``..'' for the previous (parent) directory.

```
% cd ..
```

moves you out of a subdirectory and into its parent directory.

Environment Variables

Environment variables are pieces of information used by the shell and other programs. A very important one is the PATH variable mentioned earlier. Other important variables you can set include:

- EDITOR
- TERM
- MAIL

To see what environment variables are set and what they are set to, type the command `printenv`. To set a variable, use the `setenv` command as in the example below.

```
% setenv TERM vt100

% setenv EDITOR emacs
```

Many programs mention environment variables you may want to set for them in their `man` pages. Look at the `csh` man page for some of the standard ones.

History

Most shells allow ``command line editing'' of some form or another: editing one of the previous few lines you've typed in and executing the changed line. You can set the history variable to determine how many previous command lines you will have access to. `set history=40` will let you search the last 40 commands.

Repeating and Modifying the Previous Command

The simplest form of command line editing is to repeat the last command entered or repeat the last command entered with more text appended.

If the last command you typed was:

```
% ls agreen
```

Then you can repeat this command by typing:

```
% !!
```

This will return a list of files. If you saw a directory `leavenworth` in the list returned and you wanted to list the files it contained, you could do so by typing:

```
% !! leavenworth
```

If you mistype `leavenworth` as `leaveworth` you can correct it with the following command:

```
% ^leave^leaven
```

This substitutes *leaven* for *leave* in the most recently executed command. Beware: this substitutes for the *first* occurrence of *leave* only!

Repeating Commands From Further Back in History

You can type `history` at any time to get a list of all the commands remembered. This list is numbered and you can type !*number* to repeat the command associated with number. Alternatively you can type ! and a couple of letters of the command to repeat the last line starting with the characters you specify: `!ls` to repeat your last `ls` command, for example.

The .login and .cshrc Files

The `.cshrc` file is run whenever a C shell process is started. Then, if this is a login process, the `.login` file is executed. If you are using a NeXT console with a program such as Terminal, you can usually choose whether you want each new window to execute the `.login` file by making a change to your Preferences in the Terminal program's Preferences menu. By default the `.login` will get executed.

If you are using a Sun console and you have the default setup, any `xterm` windows which you start up will not execute the `.login`.

Job Control

It is very easy to do many things at once with the Unix operating system. Since programs and commands execute as independent processes you can run them in the background and continue on in the foreground with more important tasks or tasks which require keyboard entry.

For example, you could set a program running in the background while you edit a file in the foreground.

The fg and bg Commands

When you type <Ctrl>-Z (by holding down the Control key and tapping the Z key) whatever you were doing will pause. If you want the job to go away without finishing, then you should kill it with the command `kill %`. If you don't want it paused but want it to continue in the foreground -- that is, if you want it to be the primary process to which all the characters you type get delivered -- type `fg`. If you want it to continue processing in the background while you work on something else, type `bg`.

You should not use `bg` on things which accept input such as text editors or on things which display copious output like `more` or `ps`.

What to Do When You've Suspended Multiple Jobs

If you've got several processes stopped -- perhaps you are editing two files or you have multiple `telnet` or `rlogin` sessions to remote computers -- you'll need some way of telling `fg` which job you want brought to the foreground.

By default `fg` will return you to the process you most recently suspended. If you wanted to switch processes you would have to identify it by its job number. This number can be displayed with the `jobs` command. For example:

```
% jobs
[1]          Stopped      vi .login
[2]    +     Stopped      rn
[3]          Running      cc -O -g test.c
%
```

The most recently suspended job is marked with a + symbol. If you wanted to return to job one instead, you would type:

```
% fg %1
```

You can type `%1` as a shortcut.

Starting Jobs in the Background

Some jobs should start in the background and stay there -- long running compilations or programs, for example. In this case you can direct them to the background when you start them rather than after they have already begun. To start a job in the background rather than the foreground, append an `&` symbol to the end of your command.

You should always run background processes at a lower priority by using the `nice` command. Non-interactive jobs are usually very good at getting all the resources they need. Running them at a lower priority doesn't hurt them much, but it *really* helps the interactive users: people running programs that display to terminal screens or that require input from the keyboard.

The CCO Unix Cluster has an explicit policy about the proper running of CPU-intensive background jobs. Be sure to read over this information in the Unix Cluster section. You should `man nice` and `man renice` for more information about how to alter the priority of your processes.

Suspend, z and <Ctrl>-Z

Some programs provide you with special ways of suspending them. If you started another shell by using the `csh` command, you would have to use the `suspend` command to suspend it.

If you wish to suspend a `telnet` or `rlogin` session you must first get past the current login to get the attention of the `telnet` or `rlogin` program.

Use ~ (immediately after pressing a return) to get `rlogin`'s attention. <Ctrl>-Z will suspend an `rlogin` session.

Use <Ctrl>-] to get `telnet`'s attention. <Ctrl>-]z will suspend a telnet session. Watch out, though, if you are connected from a PC with through Kermit! <Ctrl>-] is Kermit's default escape sequence. You'll need to type <Ctrl>-]<Ctrl>-]z or define Kermit's escape sequence to something else.

Summary of Common and Useful Unix Commands For Files

cp

The `cp` command allows you to create a new file from an existing file. The command line format is:

```
% cp input-file-spec output-file-spec
```

where *input-file-spec* and *output-file-spec* are valid Unix file specifications. The file specifications indicate the file(s) to copy from and the file or directory to copy to. Any part of *input-file-spec* may be replaced by a wildcard symbol (* or ?) and you may specify either a filename or a directory for the *output-file-spec*. If you do not specify a directory, you should be careful that any wildcard used in the *input-file-spec* does not cause more than one file to get copied.

```
% cp new.c old.c
% cp new.* OLD (where OLD is a directory)
```

ls

The `ls` command allows the user to get a list of files in the current directory. The command line format is:

```
% ls file-spec-list
```

where *file-spec-list* is an optional parameter of zero or more Unix file specifications (separated by spaces). The file specifications supplied (if any) indicates which directories are to be listed and the files within the directories to list.

lpr

The `lpr` command tells the system that one or more files are to be printed on the default printer. If the printer is busy with another user's file, an entry will be made in the printer queue and the file will be printed after other `lpr` requests have been satisfied. The command line format is:

```
% lpr file-spec-list
```

where *file-spec-list* is one or more Unix files to be printed on the default printer. Any part of the filenames may be replaced by a wild card.

For more information about where the printers actually are and what kind of printers are available,

please see the CCO Unix Printers reference sheet, available across from the Front Desk in Jorgensen.

man

The `man` command is a tool that gives the user brief descriptions of Unix commands along with a list of all of the command flags that the command can use. To use `man`, try one of the following formats:

`% man` *command* `% man -k` *topic*

more

The `more` command will print the contents of one or more files on the user's terminal. The command line format is:

`% more` *file-spec-list*

`more` displays a page at a time, waiting for you to press the space bar at the end of each screen. At any time you may type `q` to quit or `h` to get a list of other commands that `more` understands.

mv

The `mv` command is used to move files to different names or directories. The command line syntax is:

`% mv` *old-file-spec new-file-spec*

where *old-file-spec* is the file or files to be renamed or moved. As with `cp`, if you specify multiple files, the *new-file-spec* file should be a directory. Otherwise *new-file-spec* may specify the new name of the file. Any or all of the old filenames may be replaced by a wild card to abbreviate it or to allow more than one file to be moved.

For example:

```
% mv data.dat ./research/datadat.old
```

will change the name of the file `data.dat` to `datadat.old` and place it in the subdirectory `research`. Be very careful when copying or moving multiple files.

rm

The `rm` command allows you to delete one or more files from a disk. The command line format is:

`% rm` *file-spec-list*

where *file-spec-list* is one or more Unix file specifications, separated by spaces, listing which files are to be deleted. For example:

```
% rm *.dat able.txt
```

will delete the file `able.txt` and all files in your current working directory which end in `.dat`. Getting rid of unwanted subdirectories is a little more difficult. You can delete an empty directory with the command `rmdir` *directory-name* but you cannot use `rmdir` to delete a directory that still has files in it.

To delete a directory with files in it, use `rm` with the `-r` flag (for recursive).

Beware of the `rm *` command!

INTRODUCTION

- **Linux** is a **UNIX clone** for PCs, **freely available**.
- You may install it, use it and even distribute it, provided you obey the **"Copyleft Statement"**.
- You may also modify it, as long as you clearly mark your modifications, so that the original authors are not held responsible for your errors.
- It is a very **complete** system, with many packages and utilities you find on UNIX systems.
- For instance:
  - Languages: **C, C++, Objective C, Pascal, Fortran, CLisp**,
  - Editors: **emacs, xcoral, joe, jed, vi** and some more,
  - Debugger: **gdb**,
  - the usual **UNIX tools: make, imake, awk, sed, grep, tar, gzip, man** etc.,

  - a full implementation of **X11, XView, Interview**, several **window managers** and X11-versions of tools: **xxgdb, emacs, xcoral, xterm, xman**, etc.,
  - TEX and LATEX, including LATEX2e and various document classes,
  - **full networking**, including **TCP/IP, FTP, telnet, e-mail packages, news readers** and **World-Wide Web browsers and servers**,
  - System administrator's tools: **Tcl/Tk, Perl**,
  - information and documentation: **man-pages**, but also **HOWTOs, Manuals, FAQs**,
  - and last, but not least, **source code** is provided, or **freely available** from ftp sites.
- These lectures will introduce you to the system and its capabilities, from a **practical, user's point of view**.
- We will not enter into the internal workings of Linux. The lectures on *"Principles of Real-Time Operating systems"* will look at characteristics of real-time systems, using Linux for the examples.

- Before starting to work with Linux, you **must lose** certain **bad habits** (from MS-DOS):
  - GOLDEN RULE NO. 1: **NEVER switch off your PC, NEVER!!**
  - GOLDEN RULE NO. 2: **NEVER push reset on your PC, NEVER!!**
  - GOLDEN RULE NO. 3: **NEVER hit ctrl-alt-del at the same time, NEVER!!**

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

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

Practically all **GNU packages**, also from the *"Free Software Foundation"* run on Linux.

INNOVATIONS OF UNIX

As said, Linux is a **UNIX clone**, but Why is **UNIX** so popular?

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

A UNIX system has a *kernel*, which is in direct "contact" with the hardware and which **provides services to the user programs**. The *standard commands, the compilers, editors, text processing programs*, etc., all are **user programs**.

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

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

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

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

This is also what you want to have for a **real-time system**, for reasons we'll see again later. However, this does not necessarily mean that UNIX or Linux can be considered to be true real-time systems. Extensions do exist (conforming to the *POSIX.4 standard*), which make UNIX or Linux suitable for real-time applications. More on this later.

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

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

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

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

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

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

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

stdin of a program can be **re-directed from a file**. If the program is the **shell**, then this means that the shell can execute a previously prepared **script**.

Combined with facilities built-in in the shell, such as *loops* and *if-then-else*, this makes it possible to do

very complicated things repeatedly, without the need to type each time long series of commands.

The **daemon** is yet another concept introduced by UNIX. It is sort of a *watchdog*, who sleeps most of the time, until its attention is drawn by an (un)expected event. There are daemons to watch network activity, or to "*spool*" your and your colleague's print jobs, or to back-up your disk at 1.00 a.m., etc.

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

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

FILES AND FILE MANIPULATION

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

bin	lib	root
boot	lost+found	sbin
dev	mnt	tex
dos	mnt1	tmp
etc	mnt2	usr
home	proc	var

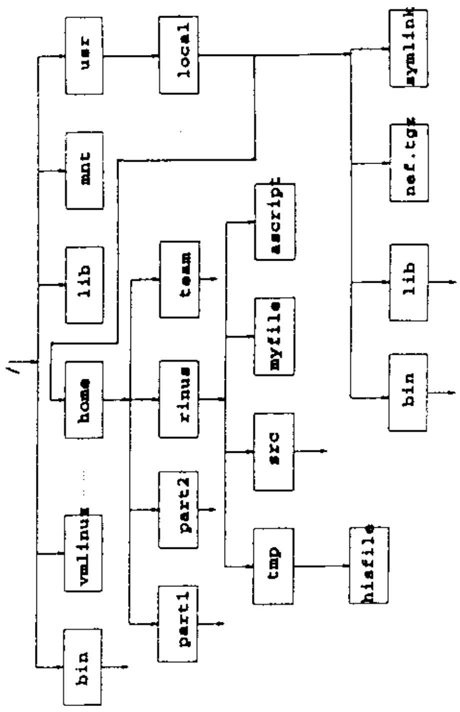
In **/usr/local/home** we find subdirectories:

part1	team
part2	rinus

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

/usr/local/home/rinus

I have a subdirectory there, called "**tmp**". From my working directory I can access a file in the directory tmp with:



relative path: `tmp/hisfile`
full path: `/usr/local/home/rinus/tmp/hisfile`

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

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

`../myfile`

and the file hisfile either as:

`hisfile`

or as `./hisfile`

. means this **current directory**
.. means the **parent** of the current **directory**.
~ means **your home directory**

You can see the names of the files in a directory with `ls`:

driver driver.tgz **exerc** exerc.tgz

ls allows many options: `ls -l` shows, among other things, also the **access rights**.

total 221

drwxr-xr-x	2	root	root	1024	Aug	10	09:25	**driver**
-rwxr-xr-x	1	root	root	9545	Aug	10	09:25	driver.tgz*
drwxr-xr-x	2	root	root	1024	Aug	10	09:25	**exerc**
-rwxr-xr-x	1	root	root	212362	Aug	10	09:25	exerc.tgz*

4

`ls -al` shows even more:

```
total 232
drwxr-xr-x  5 uli   users    1024  Aug 10 09:25  ./
drwxr-xr-x  5 root  root     1024  Aug 10 09:23  ../
-rw-r--r--  1 uli   users    3016  May 13 1994   .emacs
-rw-r--r--  1 uli   users     164  Aug  7 18:52  .kermrc
-rw-r--r--  1 uli   users      34  Jun  6 1993   .less
-rw-r--r--  1 uli   users     114  Nov 24 1993   .lessrc
drwxr-xr-x  2 uli   users    1024  Aug  7 10:28  .term/
drwxr-xr-x  2 root  root     1024  Aug 10 09:25  driver
-rwxr-xr-x  1 root  root     9545  Aug 10 09:25  driver.tgz*
drwxr-xr-x  2 root  root     1024  Aug 10 09:25  exerc
-rwxr-xr-x  1 root  root   212362  Aug 10 09:25  exerc.tgz*
```


Access rights are in the form of three letters:

r	for read
w	for write
x	for execute

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

.rw.rw.r..

The *owner* and his *group* are allowed to *read* from and to *write* to the file in this particular case, whereas *everybody else* is only allowed to *read* the file.

Directories have similar access rights.

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

```
cd ../../part1 ?
rm importantfile ?
```


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

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

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

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

There are several **commands to manipulate files and directories**:

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

For examples, see the exercises.

All files and directories you have seen so far are on the hard disk, in two **partitions**: one contains the **root filesystem**, the other is **mounted** on **/usr/local**.

Linux can access a **DOS partition** of a hard disk. The "DOS filesystem" must be **"mounted"** on the base filesystem. You could access the files with:

ls -l /dos or
ls -c /dos

You can manipulate files in /dos just as Linux files, but you **cannot execute**[1] them. Be aware of the LF vs CR-LF problem and of restrictions on filenames in DOS.

A **floppy disk** also contains a **file system of its own**. To access files on a floppy, you must **mount** its filesystem on a **mount point** of the base filesystem:

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

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

```
umount /mnt
```

GOLDEN RULE NO. 4: **NEVER pop a mounted floppy out of the drive, NEVER! !**

[1] At least not on our machines. You need dosemu for that

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

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

```
mcopy thisfile a:
```

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

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

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

SHELL

The **shell** is the **interface** between the **interactive user** and the **operating system**. It *translates* the commands the user types, into a language the system understands. The shell has also facilities built-in which make it an **interpretive language**.

The original shell was the	**Bourne shell**:	**sh**
It was followed by others:	**C shell**:	**csh**
	Korn shell:	**ksh**
	Born again shell:	**bash**

On our Linux we can choose between: **sh, ksh, tcsh** and **bash**. **tcsh** is at present the *default*, but the superuser may change it for you (*your default* is defined in the *passwd* file).

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

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

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

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

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

When you type a command, for instance

```
cat dog
```

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

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

For instance `cat fish >goldfish`

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

Input redirection: `wc <text.txt`

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

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

Careful with > if the output file exists!

For bash, 2> and 2>> redirect the standard error output.

What does > do when the file already exists ?

There are other important special characters recognised by the shell:

#	starts a **comment** if it is the first character on a line. The line is ignored by the shell.
;	**separates two commands**, which will be executed one after the other.
(and)	**group commands** together.
|	indicates a **pipe**.
*	**"expands" into zero or more characters**
?	**"expands" into exactly one** character.

*** and ?** are the so-called **wildcard characters.**

Example: `cat fish | wc >fishcount`
What will happen here? And what will happen if I type:

```
cat fi*
```

or if I say: `cat fis?`

When the shell receives a command such as

```
cat fish >goldfish
```

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

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

If I had typed:

```
cat fish >goldfish &
```

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

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

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

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

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

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

If you are interested, read a book on shell programming, for instance: Stephen G. Kochan and Patrick H. Wood; *UNIX Shell Programming* (see bibliography).

EDITORS

Several editors are available under Linux:

vi	the **classical UNIX editor**. Good for the addicts; a pain for normal users.
emacs	very complete. Two versions are available: **one runs under X11**, the other does not. The X11 version allows you to have various windows open and do debugging and editing at the same time. It has extensive help facilities.
jed	jed knows something about the **syntax of C**, and so does indenting for you, displays keywords in another colour than variables, etc
joe	a **simple, but sufficiently powerful editor**. It has a help facility and emacs key-bindings.
xcoral	Is another, very nice editor.

You invoke **emacs** with: `emacs filename` and you start typing.

If you get stuck just click on the menu item **"help"**.

The **cursor keys** do what you expect.

Move forward a whole page with: ∧v,
This means: *hold down the* **"control"** *key and hit* **v**.

backward a whole page with: **esc v** (type **ESC** and then type **v**).

Move to end of file with: esc > and to **start of file** with: **esc <**.

Use < −− and **Delete** keys for deleting.

You do an **incremental search** with ∧**s** (don't type a CR after the search pattern!).

Search and Replace with: **esc %**. Then type pattern to search for, and then pattern to replace with. It will ask you for confirmation.

X11

X11 (really *X-windows, version 11*) is a window system for UNIX machines, originally developed at MIT. It is now extended and maintained by the *X11 Consortium* of a number of major companies. At present we are at *release 6*, called **X11R6**. X11 has been ported to Linux for the Intel processors under the name *XFree86*, and is installed on your machines. Note, that although development is in the hands of commercial companies, X11 remains **free software**.

Ulrich Raich has done a splendid job in configuring X11 for our course.

Once you have installed *X11R6* on your machine, there are two ways to run it:

- **start it up manually**, with *startx*, or
- **run it automatically at boot-time**, when set up for running at *runlevel 5*.

In both cases, the start-up file, either *~/.xinitrc* or *~/.Xsession*, determines what will happen exactly.

The */etc/XF86Config* file specifies how your **graphics chip/board and your monitor will behave**. *XF86Config* fixes the **screen resolution(s)** you can use, and **opens a number of windows** for you.

The windows are under the control of a **window manager**, of which a number exist, giving different *look and feel* to your screen layout. The window manager most often used for Linux is *fvwm*. Moving your mouse to the **Window Managers** button, a menu opens up. You can choose between three window managers: **fvwm, olwm** and the **Windows 95 window manager**. You can change your window manager dynamically!

Normally, you will find at least one **"window" open**, usually an **Xterm window**, which emulates a terminal and which enables you to start working.

The background screen is called the **root window** and has usually some uniform colour. Clicking the **left mouse button**[2] in the **root window** you *pop-up* a

[2] In the case of fvwm; for olwm click the right button and, for Windows 95 use the "start" button of the menu.

menu, allowing you to choose between a number of possibilities:

- **Terminal Emulators**: open more **shells**
- **Editors**
- **Doc Readers**: reading *man pages* is one possibility
- **Graphics**: to view various things, for example, a large collection of icons
- **X Applications**: calculator, spread sheet, etc.
- **System Management**: not really for pedestrians
- **Math Applications**: two *freeware* applications are available
- **Lock Screen**: a collection of screensavers
- **xfm**: this is the **file manager**
- **Modules**
- **Exit fvwm**

When you have chosen one of those, you are again confronted with a very large choice. For instance, if you choose "Editors", you pop-up a new menu, with: *Xcoral, Emacs, jed, vi, joe, e, lyx, thot*. All this can be configured at will, but at the expense of some effort.

On the right-hand side of your screen you have an **icon box** which provides a short-cut for the applications most often used. Click *only once* on the desired icon!

You can **move windows, resize them, iconize them** and transform an **icon** back into the appropriate window, etc.

In a corner of the screen you have a view of your **"virtual desktop"**, which is much larger than your screen (4 times larger). You can move from screen to screen with a **mouse click**, or with a *CTRL-arrow key combination*.

The **X11 File Manager, xfm** merits special mention. It opens two windows for you:

- A window which displays directories and files in the usual Macintosh and Windows fashion,
- and a window displaying icons for *various types of applications*. Clicking, for instance on the icon **"Development"**, a new window appears, showing icons for **compile, link, make,** exec and **xxgdb**.

Now *drag a C source file onto the compile icon* and observe what happens!

Don't forget to "Quit" the Session before going home!

You may also make use of the **facilities** offered by *X11R6* **in your own programs**, in order to provide a **friendly interface** to the user.

You will learn more about programming for *X11R6* next week, in the lectures by *Ulrich Raich*.

There are many books on X11, see the bibliography.

Before reading the 10000 pages or so, enjoy!

Language Processors

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

gcc has literally **hundreds of options**. A number of them are used by default, so don't bother too much. Nevertheless, you should know about a few options.

gcc will produce an executable file in the current directory, with the basename of the source file, e.g. *myfile.c* will produce *myfile*. If you don't like this, either rename (mv) the file, or use the **-o option** of gcc:

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

The **-g option** produces information for the debugger. We strongly **recommend you use it** routinely. It appends to the object code all the information necessary to run the **debugger** on your program.

gcc will search certain directories for **include files (header files)**. You may specify **additional directories** to search with the **-I option**:

```
gcc -g -I ~/include -o myfile myfile.c
```

Similarly, additional **libraries** to be searched during the **link stage**, can be specified:

```
gcc -g -I ~/heads -L~/special -lpthread -o
      myapp myapp.c driver.c timer.s
```

The directory ~/**special** will be added to the *search path* and the library **libpthread.so.5** will be searched for functions that need to be linked in with the rest. Note the difference between what you specify and what you get. Standard libraries are searched automatically and need not to be specified.

Note that in the last example I have specified more than one file to be *"compiled"*. The files with extension **.c** will in fact be compiled into **assembly code**, (extension **.s**) first. Then the programs in assembly code will be translated into **relocatable object code** (extension **.o**) and finally everything will be **linked together and with the libraries** to produce the **executable program**. The compiler decides what has to be done with each file; this judgment is based on the *file extension*.

There are a few general options, which tell the compiler how far it should go in the *preprocess–compile–assemble–link chain*:

-E do **preprocessing** only.
-S stop after **compilation**, do not **assemble**.
-c **compile and/or assemble, do not link**.

You can, if you so wish, run each compiler stage separately by invoking *cpp, cc1, as and ld* respectively and specifying each time the input and output files.

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

You can **debug your program**, using **gdb**, the **GNU debugger**. **gdb** allows you to *step* through your program, to *set breakpoints*, inspect the *program stack*, *inspect and change variables*, *dump memory contents*, etc. **gdb** is a **symbolic debugger**, that is, you use the **variable names you defined in your program**

and you do not need any knowledge about **memory addresses**. See the *man page* for details.

When you work under X11, then the debugger is **xxgdb**. xxgdb has the **same functionality** as gdb, but uses a **menu- and mouse-driven user interface**. Invoking xxgdb (by clicking once on the appropiate icon) will open three windows:

- A window where to type your *commands*
- A window where the *program listing* will appear,
- And a window where gdb will display the *information you asked for*, concerning machine registers, values of variables, etc.

You can specify what you want to display about the **status** of your program when a **breakpoint** is reached. You tell gdb:
display [/f] *expr*, where /f specifies a format: hexadecimal, decimal, octal, etc. *expr* can be a variable, or an expression involving several variables.

Inspection of *stack contents*, *local or global variables* and *function arguments* can be done with the various **info** commands. The **show** commands tell things about the state of gdb itself instead of giving information about the program you are debugging.

Another feature of gdb is the **watchpoint**: the program you are debugging will stop whenever the value of a variable you defined as a *watchpoint* changes. You can then inspect the state of your program, continue execution, or make some changes to memory or values of variables, etc.

For more details, you should consult the *handout*.

Translators are available for other languages:

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

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

MAKE

One of the tools needs special mention: *make*.

make is a **command generator**.

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

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

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

If *file A* depends on *file B* and *B* was modified after *A* had been built, then *A* must be *re-built: compiled, linked, edited, substituted in a library or what have you.*

Generally you invoke *make* by typing:


```
make myprogram
```

myprogram is the **target**, it is built from one or more files, the **prerequisites** or **dependencies**. The **dependencies** are specified in a **description file (Makefile or makefile)**, together with the **commands** to be executed to **build the target**.

Example of a *Makefile*:

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

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

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

A **macro** is defined as:

```
name = a text string
```

You **refer to a macro** as:

```
$(name) or ${name}
```

Example:

```
LIB = -lX11
objs = drawable.o plot_points.o root_data.o
CC = /usr/bin/gcc
23 = "This is the 23rd run"
OPT =          # empty now, use later
DB = -g
BINDIR = /usr/local/bin
plot : ${objs}
    ${CC} -o plot ${DB} ${OPT} ${objs} ${LIB}
    mv plot ${BINDIR}
```

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


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

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

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

DIR = /usr/proj and you have **exported** it:
export DIR (for *bash*) or:
setenv DIR /usr/proj (for *tcsh*)

then you may use *${DIR}* in your Makefile:

```
SRC = ${DIR}/src
myprog : ....
    cd ${SRC}
```

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

There are special **internal macros**:

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

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

```
.SUFFIXES : .o .c .s
.c.o :
    ${CC} ${CFLAGS} -o $@ $<
.s.o :
    ${AS} ${ASFLAGS} -o $@ $<
```

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

Our first example now becomes:

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

program : ${OBJS} ${LIB}
    ${CC} -o $@ ${OBJS} ${LIB}
```


These are the **very essentials only** of *make*. For the details (many!), see the *man pages* or the book by A.Oram and S. Talbott, *Managing projects with make* (see the bibliography). The examples were taken from this book.

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

PROCESSES

Linux is a **time-sharing system**, with possibly **several users**, each running **several programs**.

The system itself has also a number of **processes** running **concurrently**: *daemons, init, X11-server* and other.

The command ps will show what is happening on your account:

PID	TTY	STAT	TIME	COMMAND
62	v06	S	0:00	/sbin/getty tty6 38400 console
88	pp1	S	0:00	-bash
98	pp2	S	0:00	-bash
104	pp2	S	0:00	xload
147	pp3	S	0:00	-bash
153	pp3	S	0:04	xdvi pract.dvi
154	pp3	S	0:06	gs -sDEVICE=x11 -dNOPAUSE -q
159	pp1	S	0:00	joe prac9504.tex
167	pp2	R	0:00	ps

ps -aux gives a lot more details:

USER	PID	%CPU	%MEM	SIZE	RSS	TTY	STAT	START	TIME	COMMAND
rinus	72	0.0	1.6	348	376	?	S	11:58	0:00	sh /usr/X11R6/lib/X11/x
rinus	82	0.1	2.8	217	652	?	S	11:58	0:00	fvwm
rinus	84	0.3	4.8	348	1116	?	S	11:58	0:01	xman
rinus	87	0.0	2.0	365	472	pp1	S	11:59	0:00	-bash
rinus	95	0.0	2.0	365	472	pp2	S	11:59	0:00	-bash
rinus	101	0.0	3.5	134	832	pp2	S	11:59	0:00	xload
rinus	105	0.0	2.2	453	528	pp1	S	12:00	0:00	joe prac9504.tex
rinus	108	0.0	0.9	85	212	pp2	R	12:05	0:00	ps -aux
root	1	0.0	0.8	44	208	?	S	11:58	0:00	init
root	5	0.0	0.5	24	124	?	S	11:58	0:00	bdflush (daemon)
root	6	0.0	0.5	24	128	?	S	11:58	0:00	update (bdflush)
root	23	0.0	0.9	56	212	?	S	11:58	0:00	/usr/sbin/crond -l10
root	35	0.0	1.0	61	236	?	S	11:58	0:00	/usr/sbin/syslogd
root	37	0.0	0.8	36	200	?	S	11:58	0:00	/usr/sbin/klogd
root	39	0.0	0.9	68	216	?	S	11:58	0:00	/usr/sbin/inetd
root	41	0.0	0.8	64	204	?	S	11:58	0:00	/usr/sbin/lpd
root	60	0.0	1.0	88	232	v06	S	11:58	0:00	/sbin/getty tty6 38400
root	61	0.0	1.8	129	424	?	S	11:58	0:00	/usr/X11/bin/xdm-nodae
root	63	2.2	7.1	2557	1660	?	R	11:58	0:09	/usr/X11R6/bin/X-auth
root	64	0.0	4.1	173	972	?	S	11:58	0:00	-0
root	69	0.0	3.8	108	896	?	S	11:58	0:00	xconsole -geometry 480x
root	86	0.1	4.7	388	1092	?	S	11:59	0:00	/usr/bin/X11/color_xter
root	94	0.0	4.7	388	1096	?	S	11:59	0:00	/usr/bin/X11/color_xter

MAN PAGES

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

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

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

Format: *command options arguments*

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

Options usually are a single character, preceeded by a - (minus sign). Several **one-character options** may be combined into a **single string**.

Sometimes, an option must be followed by an **argument**, which can be a **string** or a **number**.

Here is an example (*chown*):

The man page says:
chown [-Rcfv][–recursive][–changes][–help][–version][–silent][–quiet][–verbose][usr][:.][group]file...

The **usage message**, in this case obtained with `chown --help` produces:

Usage: chown [OPTION]... OWNER[.[GROUP]] FILE...
or: chown [OPTION]... .[GROUP] FILE...

It is perfectly legal to type on the command line:

```
chown -c -f -R .users file1 file2 file3
```

```
or: chown -cfR rinus.users file*
```

Other examples: `gcc -o outfile infile.c`

```
dvips -p5 -l9 pract.dvi
```


VIRTUAL SCREENS

You have more than one **"virtual screen"** at your disposal. You change between them with: `alt-Fk`

When you go for the first time to, say, virtual screen 3, you must **login** again there, possibly with **another user id**.

This is convenient: you login as **root** on one screen and with your own **user id** on another, where you do your normal work. This gives some protection against your own mistakes.

The virtual screen facility does **ALSO** work if you are running **X-windows, version 11**, commonly known as **X11**. Use: *CTRL-ALT-F4* and *CTRL-ALT-F7*.

X-windows is really much more **convenient** than virtual screens, but for a *critical real-time application* you may be obliged to work without X11.

UNIX TOOLS

The majority of the UNIX tools are included in most of the Linux distributions, **in executable** form.

Source code for all of the packages (also those which may not already be included in your version of Linux) is available from the **Free Software Foundation**, via ftp from one of the **archive sites**.

The UNIX Tools are usually distributed in the form of a **compressed archive** (with a filename *xxx.tar.gz* or *xxx.tgz*), containing *source code, header file(s), man pages, documentation and a Makefile*.

If you want to add a package to your system, then you will have to compile it. You will do this using *make* and the *Makefile* provided with the source code of the package.

The following standard UNIX tools are probably already part of your Linux installation:

sed	the **stream editor**
grep	for **string search**
find	to **find files**, specifying various **attributes**
tar	**archive facility**
gcc	the **C compiler**
gdb	the **C debugger**
gzip, gunzip	**file compression/decompression**
m4	**macro processor**
yacc	**yet another compiler compiler**

On-line Documentation and Text Processing

Linux comes with a vast amount of **on-line documentation** which is not necessarily all available on our present installation.

Some of the documents are in a format which can be used for **on-line browsing** (for example, using **emacs**) on the one hand and for **formatting** by LaTeX2e on the other hand (the files with the **.txi** extension).

The **man pages** can also be formatted (with **groff**, the GNU version of **troff**), or else exist in the form of (*compressed*)**ASCII files**, ready for printing or viewing.

There is also a large amount of **HOWTOs**. The total number is at present 35. Examples are: **Linux Ethernet HOWTO**, **Installation HOWTO**, **Keyboard HOWTO**, **Electronic Mail HOWTO**, **SCSI HOWTO**, **XFree86 HOWTO** etc., etc.

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

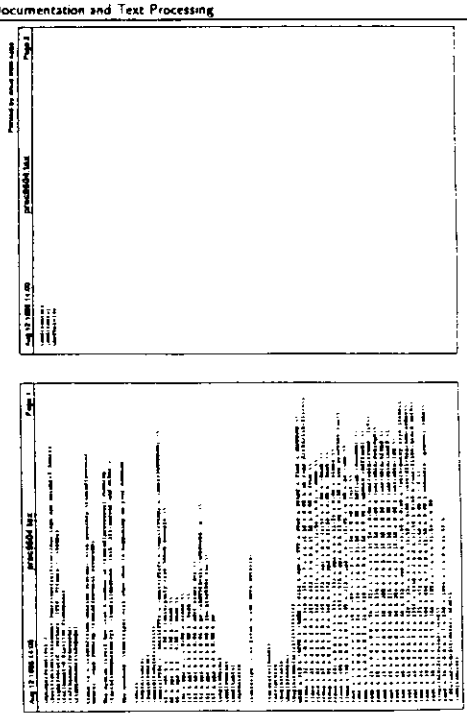
Then there exist a general **Linux Installation and Getting Started Guide**, a **Network Administrator's Guide** and a **Kernel Hacker's Guide**.

If all this is not available on your system (usually because of limited disk space), then you can find it all on the various **ftp sites**. If the file(s) you find there are in TeX format, than you can format them using LaTeX2e (or an older version of LaTeX) and **prepare for printing** with *dvips* or **view** them with *xdvi*.

xdvi makes use of *Ghostscript*, alias *gs*. Ghostscript can also be used to make a **Postscript** file **printable** on a **non-Postscript printer**.

The present transparencies were prepared with LaTeX2e, using the **documentclass** FoilTeX.

Conversely, programs exist to **prepare a normal ASCII file for printing on a Postscript printer**. *a2ps* is an example, but there are others. A sample of *a2ps output* follows.



NETWORKING

Our machines should be connected to an **Ethernet segment**. Such a segment may in turn be connected to other Ethernet segments and to a Computer Centre.

The Computer Centre may in turn be connected to the **Internet**.

Linux has **full networking support**. Drivers exist for a large number of **Ethernet boards** and usually the **kernel includes a number of those drivers**. Connecting a Linux machine to a **local area network** is therefore rarely a problem.

Connecting a machine running Linux to the **Internet** does not present significant technical problems either.

Linux comes with a **number of packages** which build on the **network connection**:

ftp	for **file transfer** via the network.
nfs	for **mounting remote filesystems**.
telnet	for **logging in to a remote machine** with your Linux machine as terminal.
talk	to talk to your friend.
ppp	to establish a **serial link** to a machine on the Internet and **become yourself an Internet node**.
slip, cslip	to do the same, but using another protocol.
term	to establish a **serial link** to a machine on the network, but without yourself becoming an Internet node.
rn, tin, etc.	various **newsreaders** and **newsservers**.
elm, pine	various **e-mail packages**.
Mosaic or Netscape	sophisticated graphical **browsers** for the **World-Wide Web**.
Lynx	a text-only browser for WWW.
www1	**CERN's WWW-server**. To turn your Linux machine into a WWW-server. e.g. to inform about your University.

To make all these services and applications work is in general a matter of *installing the necessary packages and editing a few configuration files.*

Documentation is available to help you accomplish the latter task. In particular the excellent book by Olaf Kirch, *LINUX Network Administrator's Guide* (see bibliography).

Summary of UNIX commands

<rahardjo@iscom.com>

This is a summary of UNIX commands available on most UNIX systems. Depending on the configuration, some of the commands may be unavailable on your site. These commands may be a commercial program, freeware or public domain program that must be installed separately, or probably just not in your search path. Check your local documentation or manual pages for more details (e.g. man programname).

This reference card, obviously, cannot describe all UNIX commands in details, but instead I picked commands that are useful and interesting from a user's point of view.

Most of the commands described in this reference card are explained in my UNIX and Internet book which is written in "Bahasa Indonesia" (Indonesian language.)

Rahardjo, Budi (1994). *Buku Pegangan Sistem Unix dan Internet*. Bellingham, WA: OpenPathways, xiv+251pp. Comb-bound, ISBN 1-885130-11-2.

This book is available only by mail order. Send inquiry to Marina Schneider
mschneid@pr.mail.net.org

Disclaimer

The author and publisher make no warranty of any kind, expressed or implied, including the warranties of merchantability or fitness for a particular purpose, with regard to the use of commands contained in this reference card. This reference card is provided "as is". The author and publisher shall not be liable for damage in connection with, or arising out of the furnishing, performance, or the use of these commands or the associated descriptions.

Conventions

bold
: represents program name

dirname
: represents directory name as an argument

filename
: represents file name as an argument

[dirname]
: optional directory name (or other optional argument) as argument. If it is no given, current directory will be used.

Table of Contents

1. Directory and file commands

bdf
: display disk space (HP-UX). See also **df**.

cat *filename*
: display the content of file *filename*

cd *[dirname]*
: change directory to *dirname*. If *dirname* is omitted, change to your home directory.

cp *source destination*
: copy file *source* into file *destination*.

df *[dirname]*
: display free disk space. If dirname is omitted, display all available disks. The output maybe in blocks or in Kbytes. Use **df -k** in Solaris 2.x

dtree
: (visually) display directory structure

du *[dirname]*

display disk usage.

less *filename*
display *filename* one screenful. A pager similar to (better than) more.

ls *[dirname]*
list the content of directory *dirname*. Options:
-a display hidden files
-l display in long format

mkdir *dirname*
make directory *dirname*

more *filename*
view file *filename* one screenfull at a time

mv *oldname newname*
rename file *oldname* to file *newname*. If *newname* is a directory, then move *oldname* into directory *newname*.

pg *filename*
view *filename* one screenfull at atime (a pager).

pwd
print working directory

rmdir *dirname*
remove directory *dirname* (if *dirname* is empty).

rm *file1 [file2 ...]*
remove files *file1*, *file2*, etc.

rm -r *dirname*
remove *dirname* recursively, removing all files and subdirectories underneath *dirname*.

xless
an X window pager (named after less)

2. Print-related commands

lp
print a file (HP-UX, Solaris 2.x)

lpq *[-Pprintername]*
query printer queue of the default printer. If *printername* is given, will query printer *printername*. (BSD, SunOS, Linux)

lpr *[-Pprintername] filename*
print *filename* (send *filename* to the default printer). If printername is given, will send to *filename* to *printername*. (BSD, SunOS, Linux)

lprm *[-Pprintername] jobnum*
remove printing job number *jubnum* from printer *printername*. (BSD, SunOS, Linux)

lpstat
check printer status (HP-UX, Solaris 2.x)

3. Miscellaneous commands

env
print or alter environment variables

hostname
display host name

man *topic*
display on-line manual on *topic*.

screen
create multiple screen with one physical screen. This program is useful if you have a text-only (e.g. vt100) terminal. Move around with *control-A*.

uname
print system name

users
display all users on-line

w
check who is doing what

which *commandname*
show the location of *commandname*

who
who is on-line on this machine

4. Process management

kill *SIGNUM PID*
Send signal *SIGNUM* to process ID *PID*, or kill (terminate) process with process ID number *PID*. For example:
kill -HUP 5555

nice *programname*
run program *programname* with lower priority (nicer to other users).

Recommended for running background processes.

ps
check processes. The options for this command depends on the version and variation of your UNIX. Check your local documentation.

top
show (continuously) what process(es) is running.

5. File archive and compression

compress *filename*
make *filename* smaller (compression). *filename* will be replaced by *filename.Z* (a .Z extension is added).

gunzip *filename.gz*
expand *filename.gz* into its original form (size) and remove the *.gz* extension. This is GNU unzip.

gzip *filename*
compress *filename* with GNU zip (gzip) and add .gz extension

mt
magnetic tape control program.

tar
combine files into one tar file, or extract files from a tar file. A tar file could be a device (magnetic tape as /dev/rst0) or a plain file.
To extract *filename.tar*
tar xvf filename.tar

To combine all files under *dirname* into *filename.tar*:
tar cvf filename.tar dirname

uncompress *filename.Z*
expand *filename.Z* into its original size and remove the .Z extension

unarj *filename.arj*
exteract files from an ARJ archive

unzip *filename.zip*
extract files from *filename.zip*. The reverse of zip command.

zip *zipname files*
create an archive file (and compress it) called *zipname.zip* which contains *files*.

MTOOLS

There is a set of commands to access MS-DOS disks on systems equipped with floppy disk. Most DOS file commands are available (prefixed with "m"): **mcopy, mdel, mdir, mmd, mtype,** etc.

6. Text editors

asWedit
HTML editor

axe
a simple X window text editor

ee
easy edit: emacs with a help menu

elvis
a vi clone

emacs
start emacs. A more extensive documentation is available on line. Reference card for GNU emacs is also available from FSF GNU emacs distribution.

jed
jed text editor

joe *[filename]*
a WordStar-like editor

nedit
a Motif-based text editor

pico
a simple text editor distributed as part of **pine**

sqhm
SoftQuad's HoTMetaL HTML editor.

textedit *[filename]*
OpenWindow's text editor

vi *[filename]*
vi editor

vile *[filename]*
a vi-clone

vim *[filename]*
a vi-clone

xcoral
a multiwindow X window text editor that can be used to browse C functions and C++ classes.

xedit
a simple X window text editor

xemacs
X window Emacs (formerly Lucid Emacs)

7. Mail Programs

biff
notify new mail has arrived

elm
read and/or compose e-mail.

fastmail
quick batch mail (part of Elm)

from
list senders of mails in your mailbox

frm *[foldername]*
similar to **from**, but has a better output. If *foldername* is present, it will list senders in folder *foldername* instead your incoming mailbox. This program is distributed as part of the **elm** package.

mail *[userid]*
read mail. If *userid* is given in the command line, it will be used in compose mode. After done, mail will be sent to *userid*

mailtool
OpenWindows mail program

mush
mail user shell. Similar to **mail**, but has a better user interface.

newmail
notify new mail has arrived (part of **elm** package)

pine
mail reader and composer.

xwafemail
X window interface of mail written in **wafe+perl**.

There are also other e-mail packages, such as MH, which come with their own commands. For example, MH has the following commands (and more):

inc
incorporate mail into inbox folder.

scan
scan mailbox/folder

show
show current selected e-mail.

emacs, for example has its own mail reader **M-x rmail** and also **M-x mh**, within emacs window.

8. Usenet news

knews
interactive X window-based news reader

nn
read news with **nn** (NoNews is good news.) Can be used to read through NNTP or spool.

nntidy
cleanup or tidy your .newsrc

rn
read news with **rn**.

slrn
NNTP-based newsreader (can display color ANSI).

tin
newsreader

trn
threaded **rn** newsreader

xrn
X window-based news reader

xvnews
Xview-based newsreader

xwafenews
X window-based newsreader written in wafe+perl

9. File transfer and remote access

bftp
batch FTP

ftp *hostname*
Using the **ftp** prgram to perform FTP to/from host *hostname*.

ftptool
X-window (xview) based FTP prgram

kermit

send or receive files with kermit protovol.
kermit -s *filename*
to send filename
kermit -r *filename*
to receive filename

minicom
communication package similar to Procomm

ncftp *hostname*
a user-friendly FTP program

rlogin *hostname [-l userid]*
remote login to host *hostname*. If *-l userid* is given, will login as *userid*

rsh *hostname*
remote shell to host *hostname*

rz
receive (upload) files with zmodem

seyon
X window communication package

sz *[-r] files*
send (download) files with zmodem

telnet *hostname [portnum]*
connect to *hostname* with telnet. If *portnum* is specified, connect to port *portnum*. (Usually *portnum* is required if you want to connect to various services such as IRC or MUD.)

xc
xcomm communication package

xftp
X-window interface to FTP

10. X window

openwin
start OpenWindows

startx
start X window

Window manager

The following programs (window managers) are usually started when you type **openwin** or **startx**. It is usually in your .xinitrc file.

fvwm
feeble window manager

mwm
Motif window manager

olvwm
OpenLook virtual window manager

olwm
OpenLook window manager

twm
tab window manager

X window programs

filemgr
OpenWindows file manager

oclock
display clock

xcalc
calculator

xclock
display clock

xfm
file manager

xlock
lock your screen

xodo
odometer, track the distance your mouse travel

xterm
a terminal or shell session

11. Graph, Plot, and Image Processing Tools

coreldraw
start CorelDraw (commercial drawing program)

corelpaint
start Corel Paint program (a commercial painting program)

ghostview
a front-end of ghostscript (gs).

giftrans
converts GIF image to transparant GIF.

gimp
image processing tool/drawing program with filters (plug-ins) to manipulate image.

gnuplot
a freeware plotting program capable of plotting 2D and 3D plots. It supports a wide variety of output formats.

gs
Ghostscript, a PostScript previewer. It can also be used to convert PostScript into other graphic formats.

pageview
preview PostScript file

ps2epsi *file.ps [file.epsi]*
create Adobe's Encapsulated PostScript Interchange (EPSI) format from a postscript file.

psselect *infile [outfile]*
select pages from a PostScript file.

pstoedit
converts PostScript to tgif format for editing

rplot
plot 2 D data

sxpm
show an XPM (X Pixmap) image.

tgif
an X window drawing tool. It can produce various output formats, including PostScript.

xfig
an X window drawing tool capable of producing fig output

xgraph
a simple X window graphing program able to produce a bar graph.

xloadimage *filename*
image previewer

xpaint *[filename]*
X window painting program, undrestands various image format including GIF

xv *[files]*
image previewer and manipulation tools for X window. It supports various formats, including GIF, BMP, TIFF, and PostScript.

12. Information Systems

archie
search the Archie database for anonymous FTP sites

arena
X-window WWW-browser which understands HTML 3.0.

chimera
X-window WWW browser

gopher
a Gopher client

hotjava
WWW browser that understands *Java* language

hypermail
converts mail into HTML

hytelnet
access various libraries on the Internet

lynx
a text-based WWW-browser

Mosaic
NCSA X window WWW-browser

netscape
a WWW browser (X window) with a built in threaded newsreader

sgopher
a simple Gopher client, supports dumb terminal

swais
a text-based WAIS client

tkwww
Tk based WWW browser

willow
a library, Z39.50 and WWW browser

xarchie
X-window interface of **archie**

xgopher
X-window gopher client

xmosaic
X-window WWW-browser

xwais
X-window WAIS client

13. Networking programs

finger *userid@hostname*

check information of *userid* at host *hostname*

host *domainname*
find information about *domainname*, such as its MX record or IP address

irc
Internet Relay Chat, a multi-user chat. Beware, addictive!

lpmudr
an LP MUD client. Beware, addictive!

nslookup
query information about a specific host through a domain name server. For example you can find IP address of a machine, MX record of a domain.

ping *hostname*
check if host *hostname* is alive

rup *[hostname]*
show status of local machines. Of *hostname* is given, only check that *hostname*

talk *useri@hostname*
talk to *userid* at host *hostname*

tf
Tiny Fugue, a MUD client

traceroute *hostname*
tracing IP packet from this host to *hostname*

write *userid*
write to *userid* screen/session

zlocate *userid*
use Zypher to locate userid (where or which machine the user logs on)

zwrite *userid*
send a personal message to *userid* through zypher.

14. Programming tools and shells

acc *[files]*
Sun's ANSI C compiler

bash
Born again shell. A **sh** clone, but better

bison
a GNU implementation of **yacc**

byacc
Berkeley yacc

cc
C compiler

CC
Sun's C++ compiler

cpp
C language preprocessor

csh
C-shell

dbx
a debugger program

dbxtool
X window interface to **dbx**

f77
FORTRAN compiler

flex
GNU implementation of lex

g77
GNU Fortran compiler

g++
GNU C++ compiler

gawk
GNU **awk**

gcc
GNU C compiler

gcl
GNU Common Lisp

gdb
GNU debugger

gofer
a Haskell implementation

grep *pattern file(s)*
search for a string, pattern, or regular expression in file(s)

imake
C preprocessor interface to the **make** command. Usually it uses file **Imakefile**.

kcl
Kyoto Common Lisp

ksh
Korn shell

lex
lexical analyzer generator

lint
verify a C program

m4
macro language processor

make
maintain, update, compile, and regenerate related files by a set of rules defined in file **Makefile**.

nm
print name list of object file

pc
pascal compiler

perl
Practical Extraction and Reporting Language, a powerful programming and scripting language

prolog
a Prolog system

rcs
resource / version control

sccs
source code control system

sed
stream editor

sh
Bourne shell

sml
New Jersey Standard ML

tclsh
Tcl shell, a Tcl interpreter

tcsh
tcshell, a csh compatible shell but better

tgrind
reformat source code to make it pretty for printout

wish
a simple windowing shell, a tk/tcl implementation

xcl
ANSI C compiler for AIX

yacc
parsing program generator. Generate C code from a grammar

zsh
Z shell

15. Text processors, typesetters, and previewers

a2ps
converts ASCII to PostScript

grodvi
convert troff to TeX DVI

groff
GNU *roff

grops
convert troff to PostScript

gtroff
GNU troff

gxditview
display gtroff output files under X window.

ispell *filename*
interactive spelling program

latex *filename.tex*
process *filename.tex* with **LaTeX** and generate *filename.dvi* (a DVI file)

lout
a layout or typesetting program

mp
format text, mail, news into a nice PostScript output.

nroff *filename*
process *filename* with **nroff**

psnup
manipulate PostScript file to have *n* pages on one side

pstops
manipulate PostScript files to produce 2-up, 4-up, booklet, rotate and many other functions.

spell *filename*
spell the contents of *filename*

tex *filename.tex*
TeX document processing and typesetting.

troff *filename*
process filename with **troff**

xdvi *filename.dvi*
preview DVI file (*filename.dvi*)

xtex *filename.dvi*
preview *filename.dvi*

16. Wordprocessors

ez
Andrew Toolkit wordprocessor (free).

imaker
start an international version of FrameMaker

maker
start FrameMaker

tps
start Interleaf

wp
start Word Perfect

17. Spreadsheets

oleo
GNU spreadsheet

sc
spreadsheet calculator

xspread
X window version of **sc**

18. Databases

isql
connect to Sybase server

jinx
a curses, perl-based database

sybperl
access Sybase server with **perl** (script).

dbflst, dbfget, dbfadd, dbfdel, dbfpack, dbfcreat, dbfscan
DBF package to access DBF-format file (usually generated by xbase program), developed by Brad Eacker <beaker@sgi.com>