Computational physics PY2050

Course Details: http://www.tcd.ie/Physics/People/Charles.Patterson/teaching/PY2050_CP

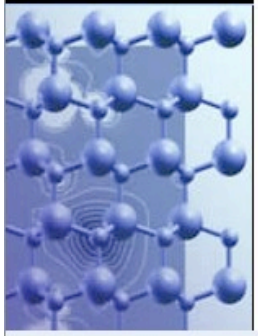
The University of Dublin | Trinity College
Ollscoil Átha Cliath | Coláiste na Tríonóide

A-Z | Search TCD go

Teaching Pages
Dr. Charles Patterson

Home Physics Contact

- research
- teaching
- people
- publications
- research opportunities
- facilities
- contact
- sitemap



http://www.tcd.ie/proxy.cgi

PY2050 Computational Physics Laboratory

Scripts and source code files for each of the three Computational Laboratories

- Brief instructions on compiling C programmes
- Brief instructions on using the vi editor are available at www.unix-manuals.com/tutorials/vi/vi-in-10-1.html
- An introduction to Programming in C
- Programme files needed for Programming in C document
- Instructions for writing up the computational physics laboratories.
- Lab 1 Finding minima of functions; root.c programme
- Lab2 The nonlinear pendulum; The pendulum.c programme
- Lab3 Fourier series; invfourier.c programme.

School of Physics Emblem

School of Physics, Trinity College Dublin, Dublin 2, Ireland.
Tel:+353 1 8961675, Fax:+353 1 6711759, e-mail: physics@tcd.ie
Contact: chris.smith@tcd.ie
Last updated: Feb 12 2007.

Course general information

6 Weeks course

Week 1 and 2 - Introduction to Linux and C programming

Week 3 and 4 - Lab1: Finding minima of functions

Week 5 and 6 - Lab2: The non-linear pendulum

Week 1: Feb 25
Week 2: March 11
Week 3: March 18
Week 4: March 25
Week 5: April 1
Week 6: April 8
Report 1 due April 1
Report 2 due April 15

Lab reports are due in one week after you have completed the lab.

- **Collect information as you do the lab**. Name files so you know what they are.
- Reports should be in the **style of a scientific document**
- Try to convey all the information in about **4-6 pages**.
- You should describe the algorithms used, however do not include the code.

Send reports to James.mcateer@tcd.ie , or hand them directly to me.

Rm 2.2, Fitzgerald Building (turn left at Sch Lecture theatre. past the entrance to the observatory and through the double doors)

Why computing?

- Theoretician: most problems can only be solved numerically
- Experimentalist: data fitting and processing

Why Unix/Linux?

- Extremely powerful and flexible computing environment
- System of choice for most super-computing centres

Why C?

- Simple but highly flexible programming language
- Maybe the most widespread programming languages

Introduction to Linux

What is an Operating System (OS)?

- The job of the of an operating system is to control the various parts of the computer, such as the cpu, memory, hard drives...

- At the same time the operating system has to provide a user interface

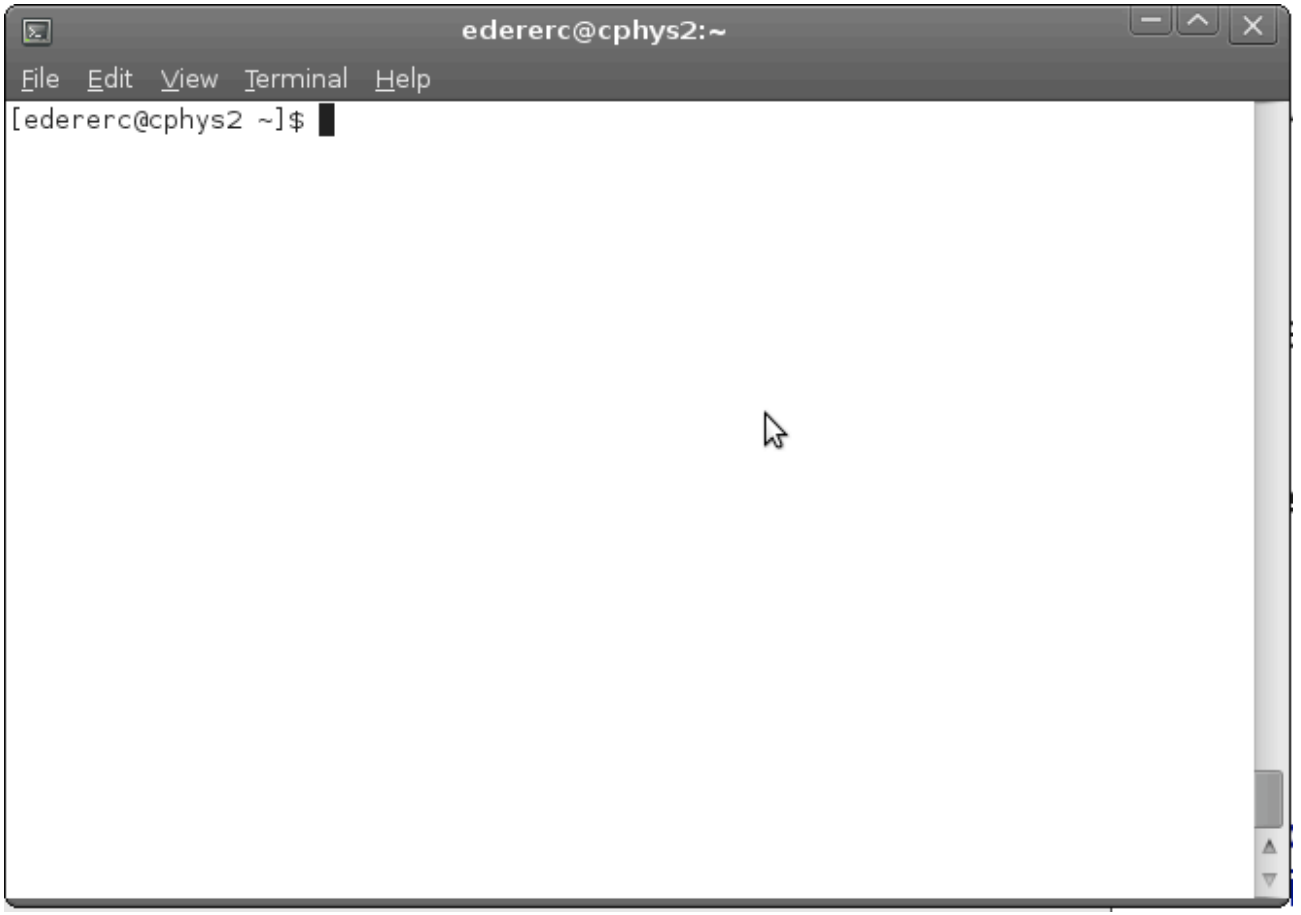
- Everything on the computer is mediated by the operating system

- Example OS: Windows 95/98/ME/NT/XP, OSX, Linux, UNIX ..

How is Linux different to Windows?

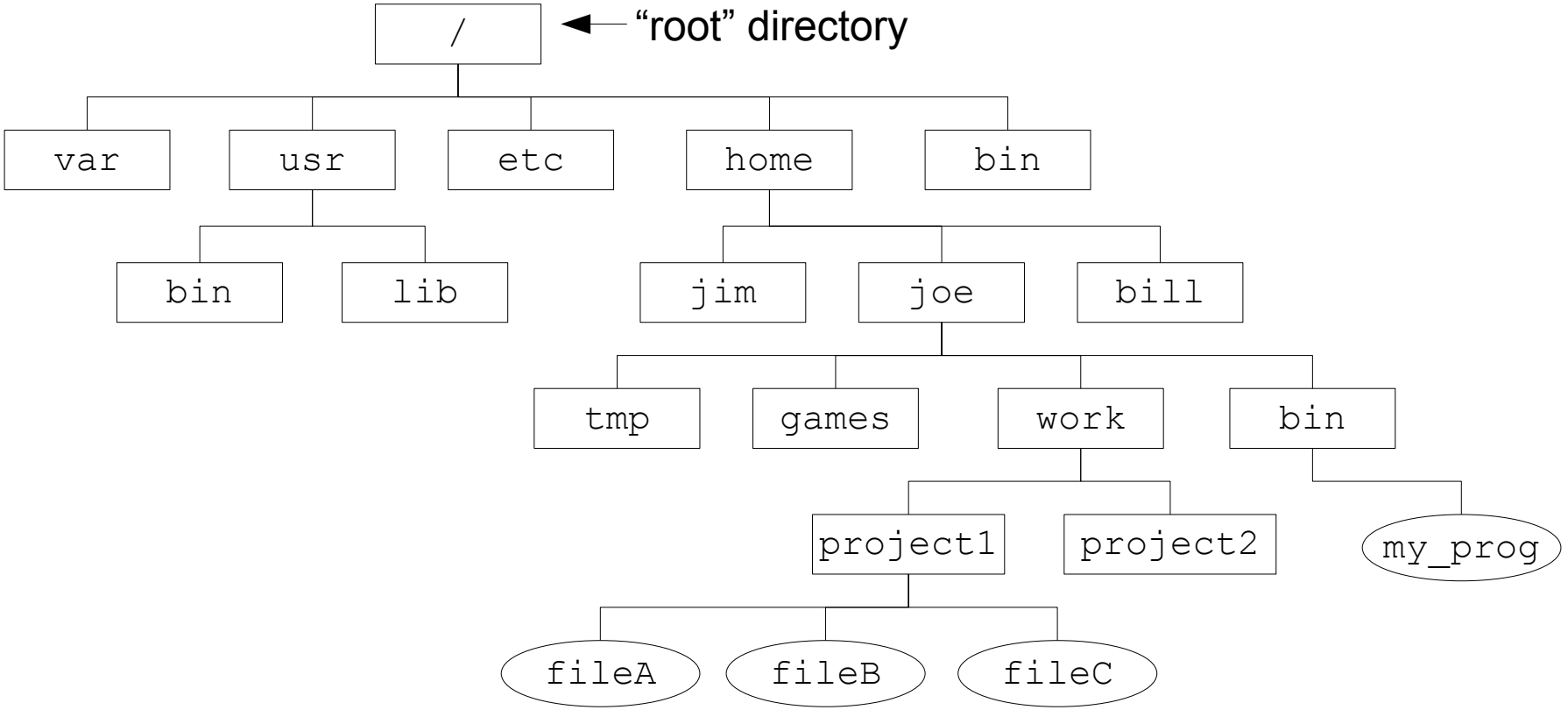
- Open source – Developers are free to edit and develop the OS
- Many flavours of Linux: Red Hat, SuSE, Ubuntu ...
- There are no Linux viruses (as of yet) so no virus checkers
- Generally Linux requires less CPU power and memory than windows
- Many (free) application programs offer similar functions as corresponding Windows programs (OpenOffice, Firefox, …)
- Linux command line is much more powerful than the windows equivalent

The terminal window



...is (almost) everything you need in this class!

Filesystem (Directory Tree)



Absolute pathnames:
`/home/joe/bin/my_prog`
`/usr/lib/`

Relative pathnames: (`pwd: /home/joe`)
`work/project1/fileA`
`../var/`

Special abbreviations:

`.`	current directory
`..`	parent directory
`~`	home directory (e.g. `/home/joe`)
`~jim`	other user's home (e.g. `/home/jim`)

Users, Groups, File Permissions

Users

`root` (super-user, system-administrator) and normal users

Groups

each user is member of at least one group (determines access to files, devices, etc).

```
ederer> ls -l
total 3
drwxr-xr-x    2 ederer  users    4096 Oct  1 11:40 projects/
-rw-r--r--    1 ederer  users    4096 May 28  2007 old_data
-rwxr-x---    1 ederer  users  227964 Sep 30 17:35 a.out
```

permissions	user	group	size	last modified	filename

File permission string: `drwxr-xr-x`

File type user(owner) group others

Filetype normal file (`-`), directory (`d`), link (`l`)

`r` **r**ead permission

`w` **w**rite permission (includes renaming and deleting)

`x` e**x**ecute - right to run program or change into directory

change permissions with `chmod`

Commands

- Many commands require specification of one or more **arguments**: `arg1 [arg2]`
- Most commands can be “fine-tuned” using **options**: `command [-o] [--long-option]`
- Sometimes options can also have arguments

```
ederer> ls -l
total 3
drwxr-xr-x    2 ederer  users    4096 Oct  1 11:40 projects/
-rw-r--r--    1 ederer  users    4096 May 28  2007 old_data
-rwxr-x---    1 ederer  users  227964 Sep 30 17:35 a.out
ederer> ls
projects  old_data  a.out
ederer> ls projects
lecture  project1  old_simulation
ederer> ls -aF --color=always
./  ../  projects/  old_data  a.out*
```

Filesystem navigation

`pwd`	- **p**rint **w**orking **d**irectory
`ls`	- **l**i**s**t (directory contents)
`cd`	- **c**hange **d**irectory (`cd` with no argument: go home, `cd ..` : go one level “up”)

Manipulation of files/directories

Important:

- Linux is case-sensitive: `file`, `File`, and `FILE` are three different files!
- There is no trash bin on the command line. Deleted files cannot be restored!

`mkdir`	–	**mak**e **dir**ectory
`rmdir`	–	**rem**ove **dir**ectory
`cp`	–	**cop**y
`mv`	–	**mov**e (rename)
`rm`	–	**rem**ove
`touch`	–	change access/modification times or creates empty file if it doesn't already exist

`mkdir -p dir1/dir2/dir3` make nested directories

`rmdir -p dir1/dir2/dir3` remove nested directories (DANGEROUS!)

`cp -i` "interactive", prompt before overwriting files

`cp -p` preserve specified attributes (default: mode, ownership, timestamps)

`cp -r` copy directories recursively

`mv file1 file2` rename '`file1`' to '`file2`'

`mv file dir` move '`file`' into '`dir`'

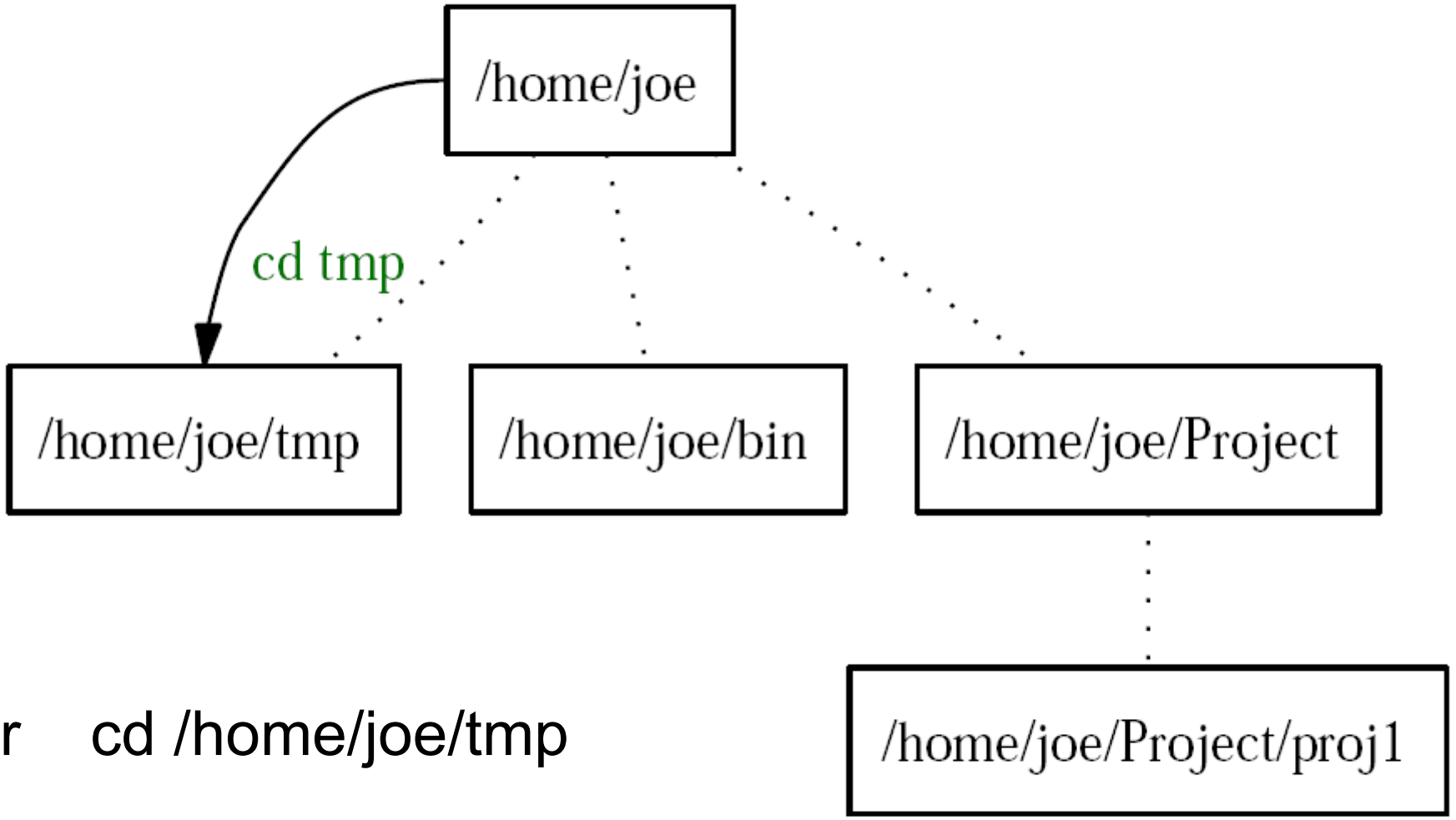
`mv dir1 dir2` if '`dir2`' exists: move '`dir1`' into '`dir2`'
if '`dir2`' does not exist: rename '`dir1`' to '`dir2`'

`mv dir file` NOT ALLOWED!

`rm file` remove '`file`'

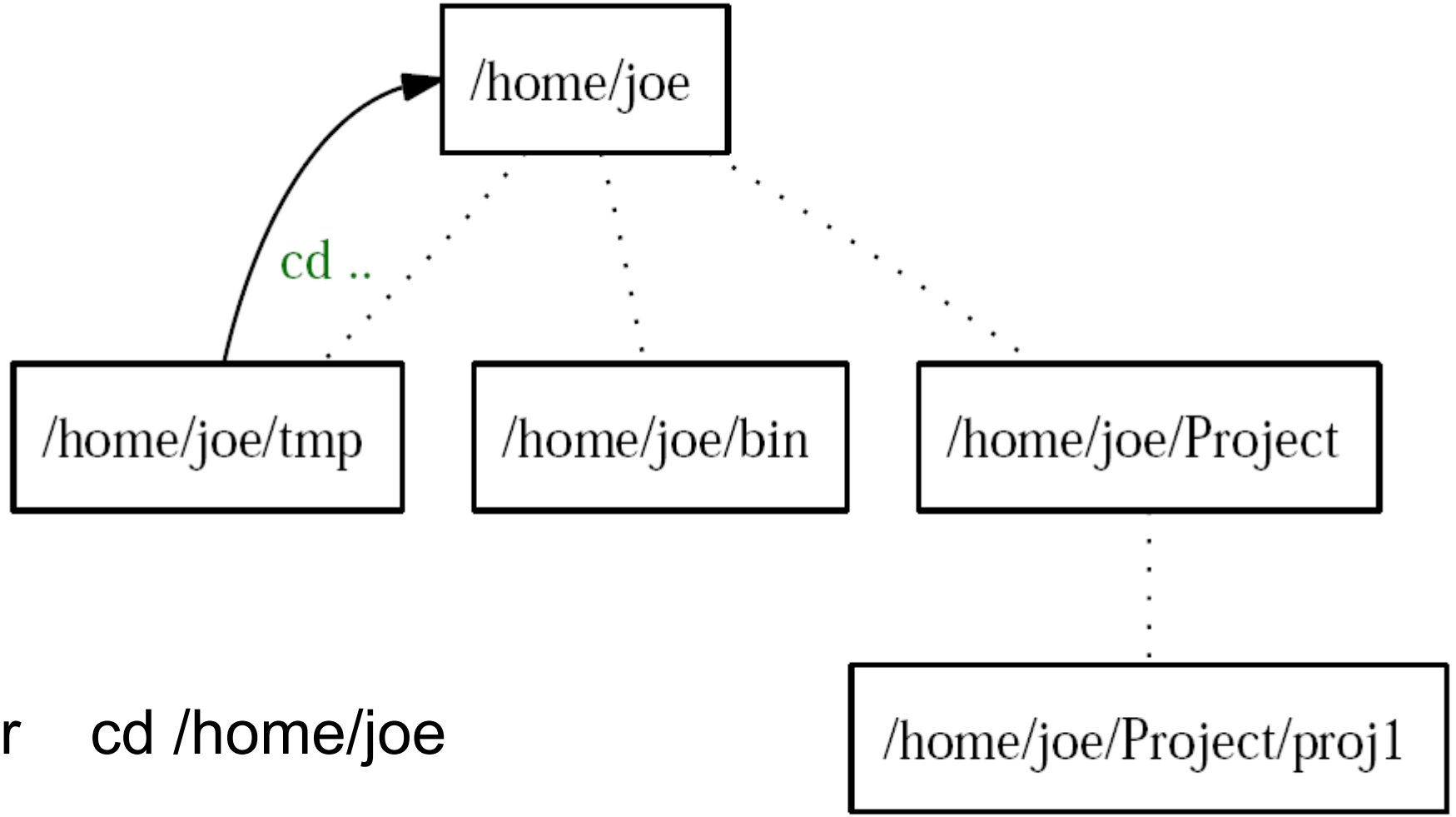
`rm -r dir` remove '`dir`' recursively, i.e. with all its contents (DANGEROUS!)

Change Directory



or cd /home/joe/tmp

Change directory



or cd /home/joe

Moving

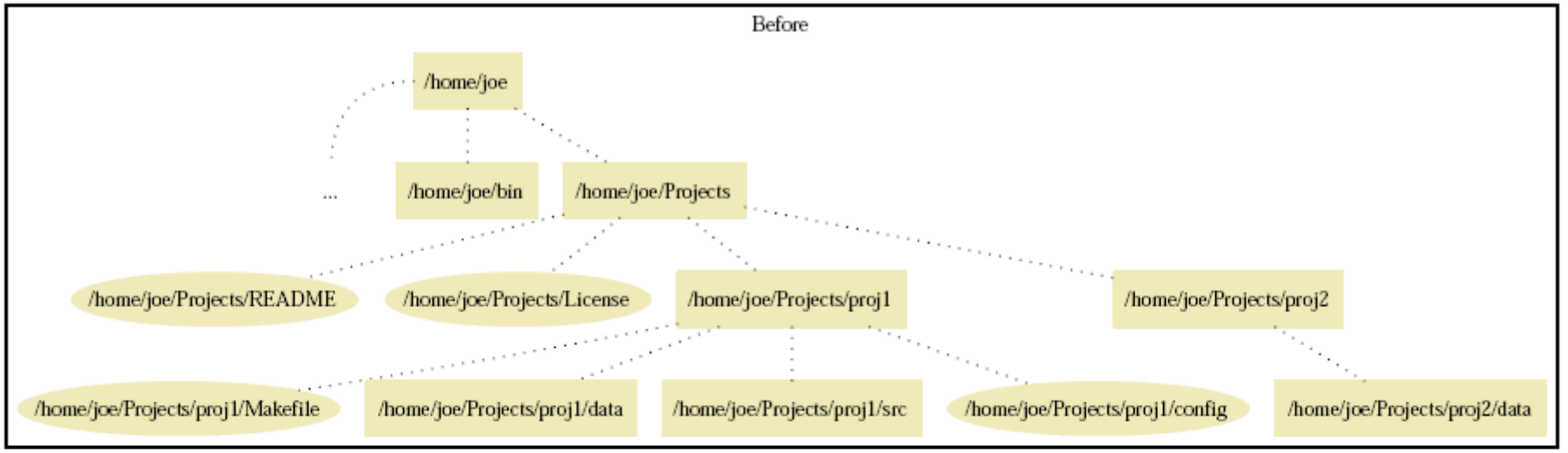
file → directory

Using `mv` with a file as a first argument and a directory as a second argument moves the file *into* the directory

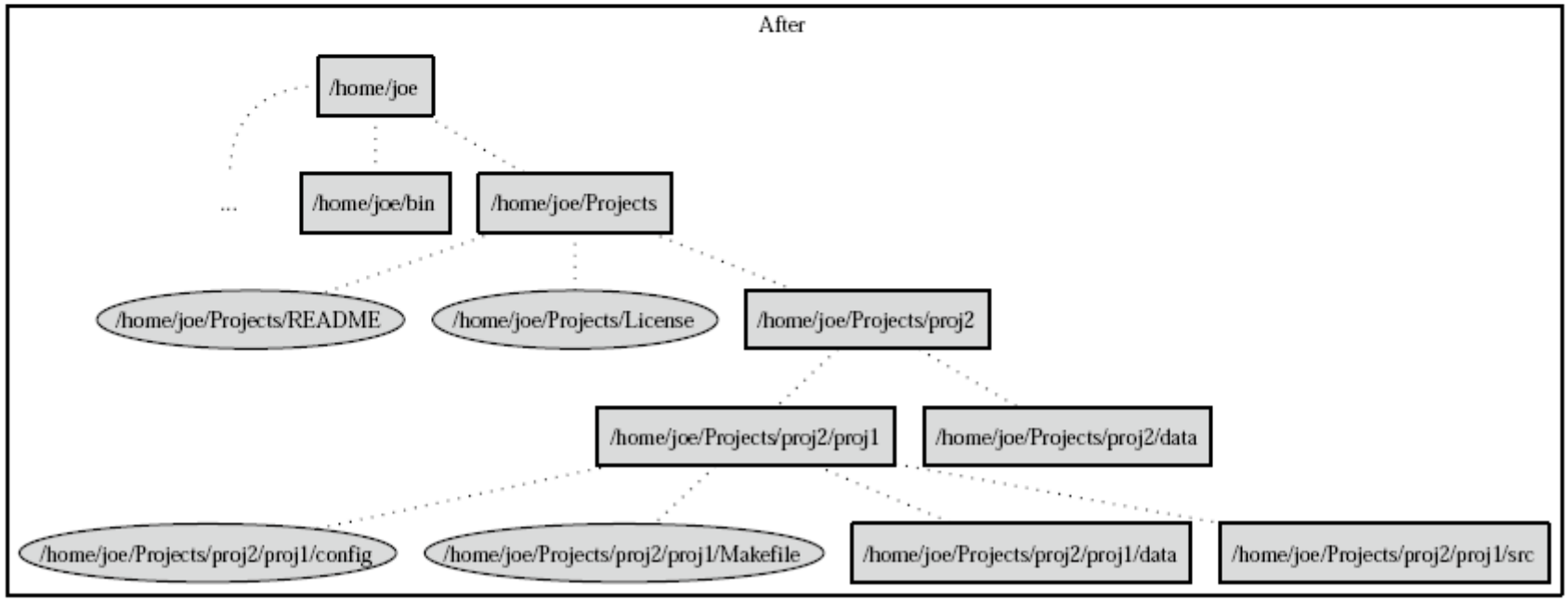
```
joe@fred$ ls
file1 file2 dir1
joe@fred$ ls dir1
file3
joe@fred$ mv file1 dir1
joe@fred$ ls # file1 is gone
file2 dir1
joe@fred$ ls dir1 # file1 moved to dir1
file1 file3
```

Move Directory

```
joe@fred$ cd /home/joe/Projects
```

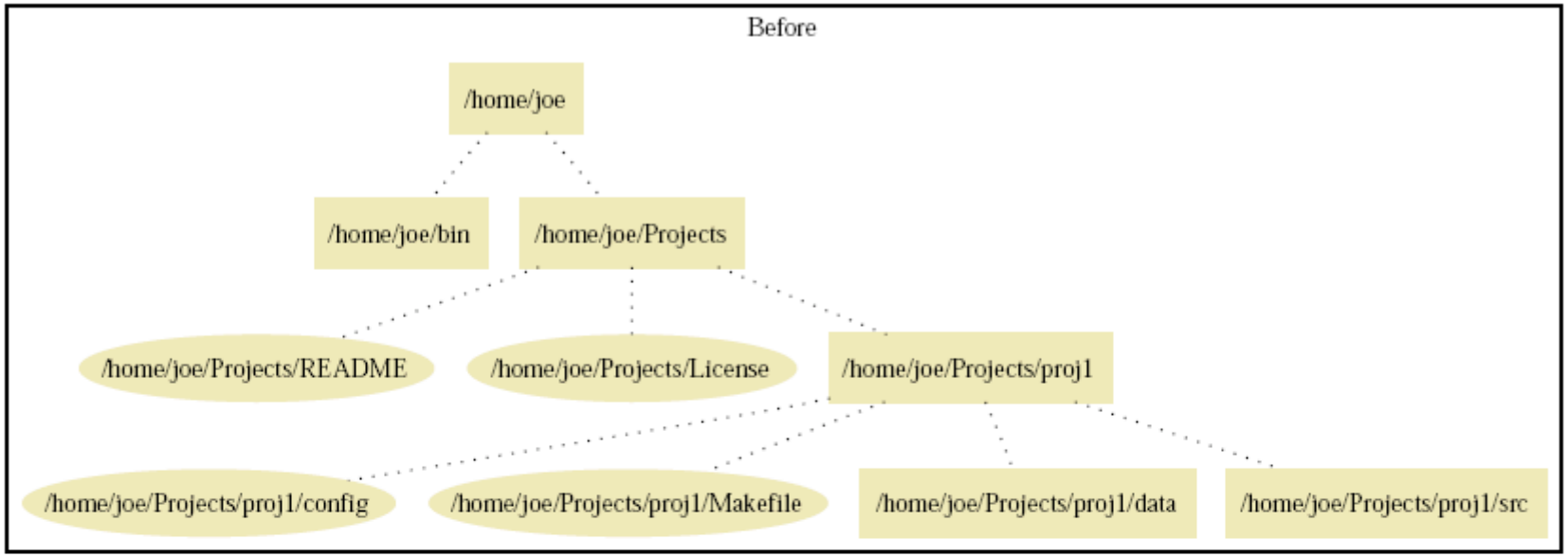



```
joe@fred$ mv proj1 proj2
```

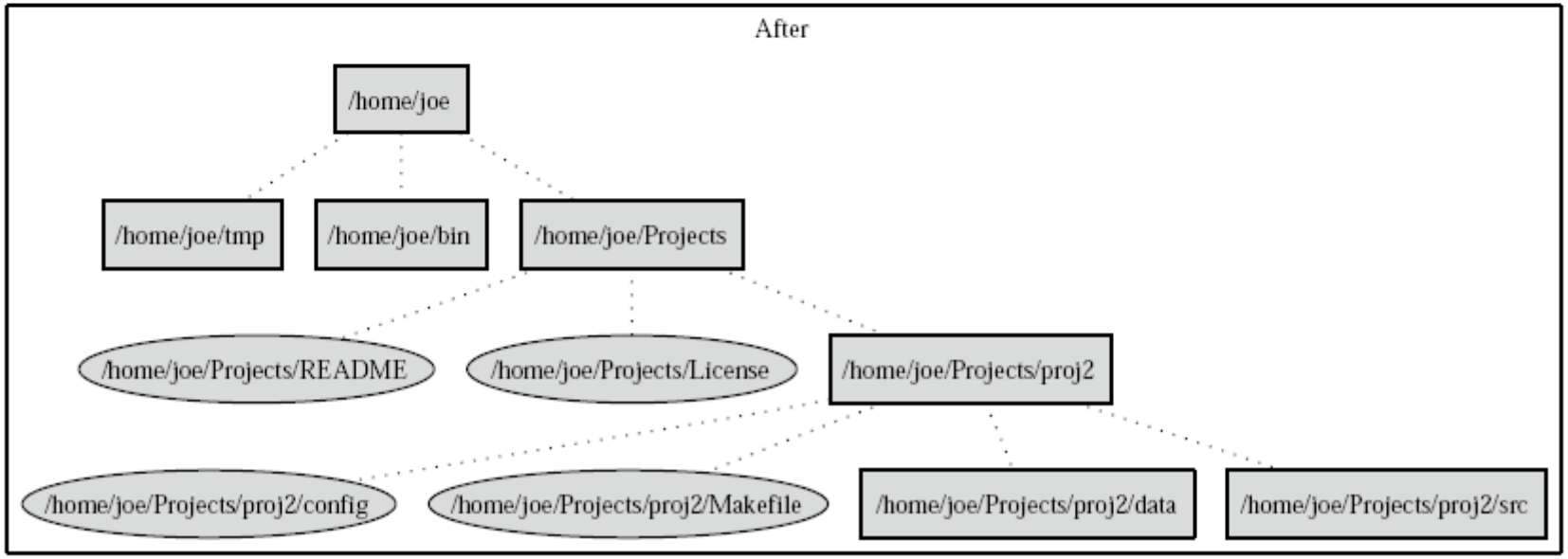


Rename directory

```
joe@fred$ cd /home/joe/Projects
```

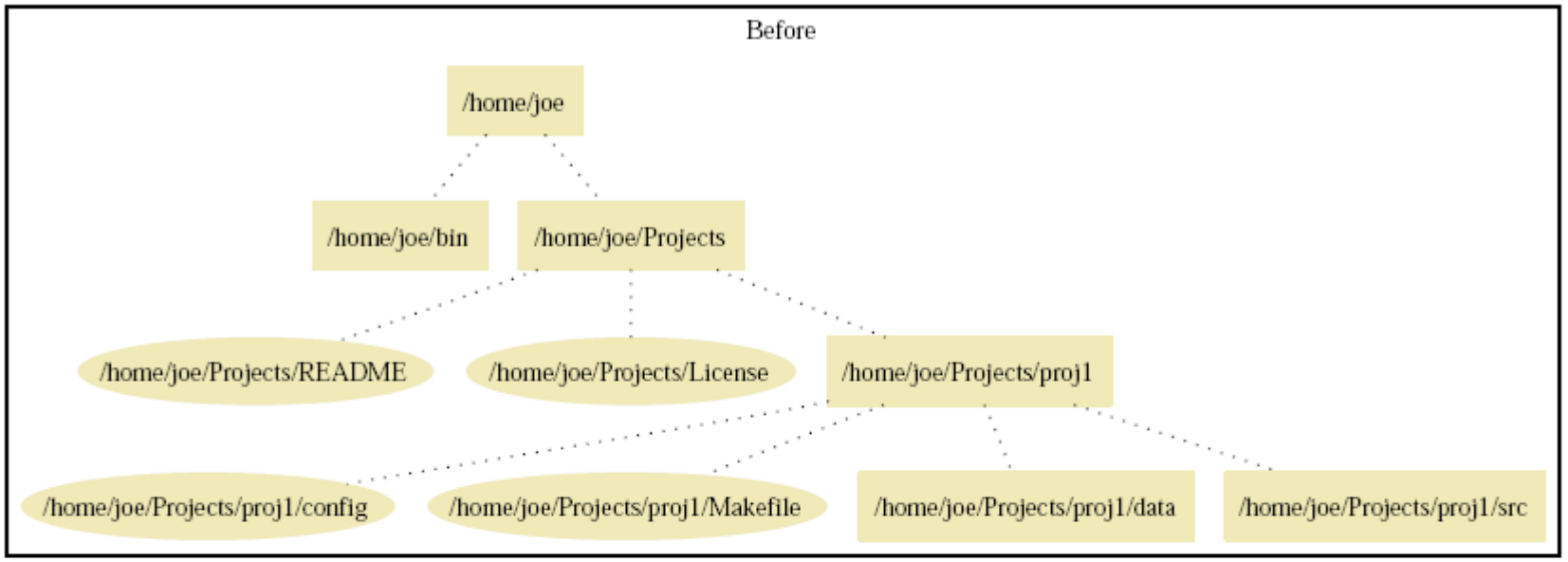



```
joe@fred$ mv proj1 proj2
```

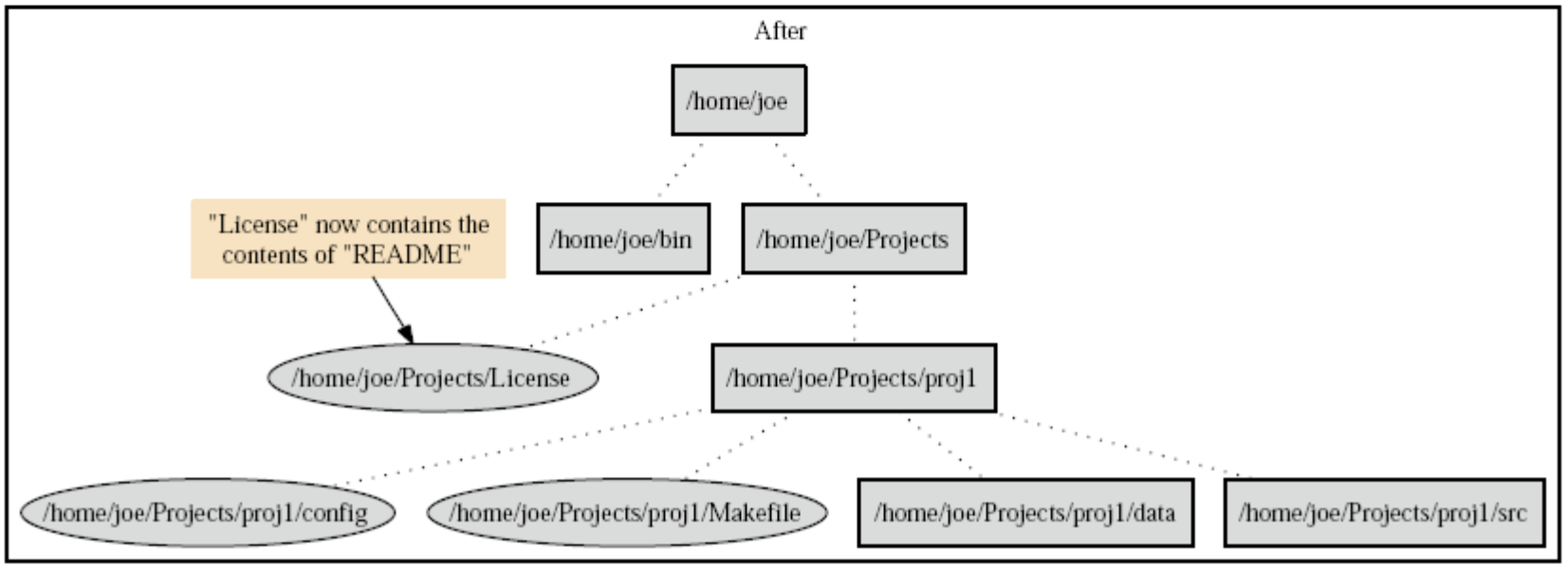


Rename

```
joe@fred$ cd /home/joe/Projects
```

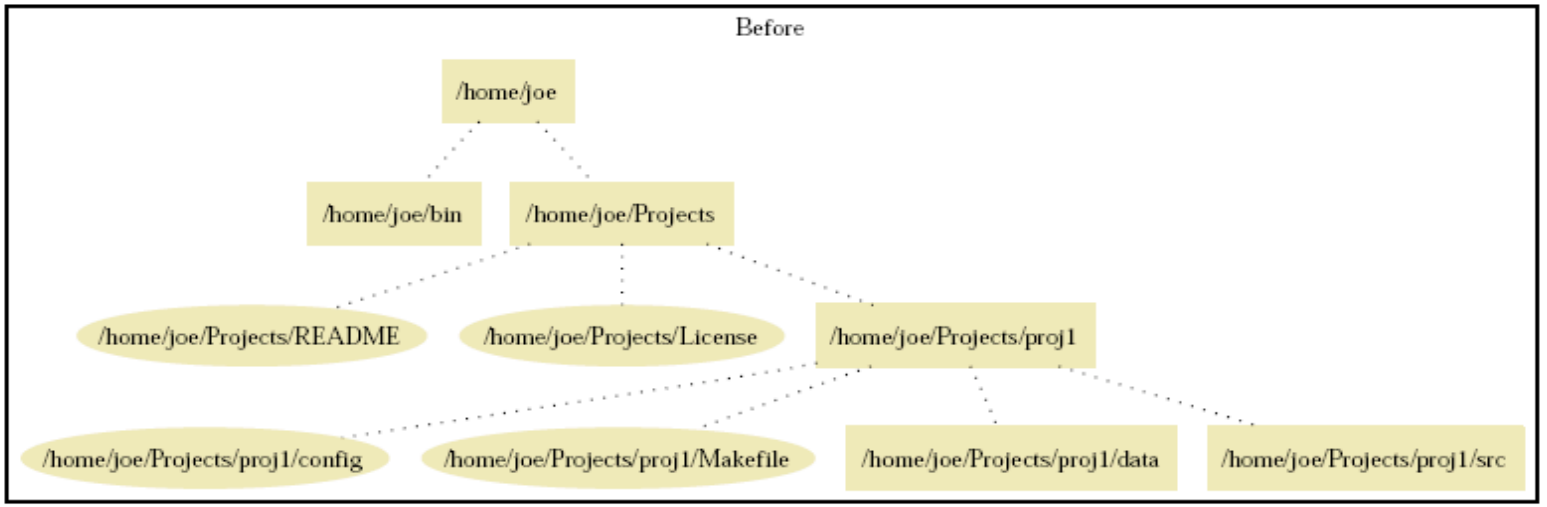



```
joe@fred$ mv README License
```

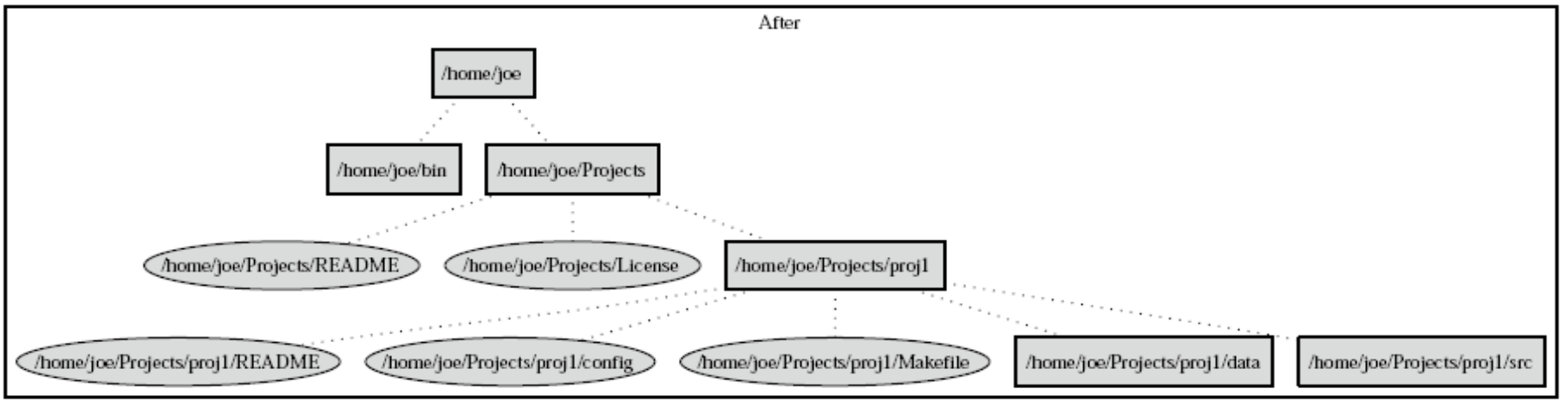


Copying

```
joe@fred$ cd /home/joe/Projects
```




```
joe@fred$ cp README proj1
```



Removing

```
joe@fred$ pwd
/home/joe
joe@fred$ ls
file1 file2 testdir
joe@fred$ rm file1
joe@fred$ ls
file2 testdir
```

- `rm` doesn't just work in the current directory alone
- It takes path arguments like `cd`

```
joe@fred$ pwd
/home/joe
joe@fred$ ls testdir
tdfile1 tdfile2
joe@fred$ rm testdir/tdfile2
joe@fred$ ls testdir
tdfile1
```

Reading/writing text files (ASCII files)

Reading

`cat`	– concatenates files and writes them to standard output (screen)
`more`	– pages through text file, one screenful at a time (hit SPACE or RETURN, `q` to quit)
`less`	– more sophisticated pager (“less is more”), can also scroll backwards

`cat -n file`	prints line numbers at the beginning of each line
`less file`	use SPACE, RETURN, PAGEUP, PAGEDOWN, or arrow-keys to navigate `g` or `<` to jump to start of file, `G` or `>` to jump to the end, `q` to quit `/pattern` to search for the string '`pattern`'; `n` to jump to next occurence

Writing

`nano [file]`	– simple terminal based text editor
`gedit [file]`	– simple GUI editor (Gnome) My recommendation for beginners!
`emacs [file]`	– complex text editor (`emacs -nw` starts terminal-based version)
`vi [file]`	– the classical unix text editor

Wildcards

Wildcards allow to handle many files with similar names simultaneously

`*` matches any string
`ls *.jpg` (list all files ending with '`.jpg`')

`?` matches any single character
`ls file?.dat` (matches `file1.dat`, `files.dat`, but not `file10.dat`)

`[abcde]` exactly one of the listed characters
`ls data[137]` (matches `data1`, `data7`, but not `data13` or `data5`)

`[2-7]` exactly one character in the given range
`ls file[a-e]` (matches `filea`, `filee`, but not `filez` or `fileaa`)

`{xy,linux,unix}` exactly any of the given strings

Help

`man`	– **man**ual page (detailed command description including all command line options)
`info`	– info pages (even more detailed description using hyperlinks)
`whatis`	– very brief description

- Most commands display a brief description when called with option `-h` or `--help`
- Plenty of help/information available on the www

Exercise 1:

1. Go to your home directory. How many files of different type are there?
2. Go one level up and list the contents of that directory
3. Go to the root directory (`/`). Inspect its contents
4. Go to the `/bin` directory and list its contents. Do you recognize some of the filenames?
5. Inspect the contents of the `/usr/bin` directorty without changing to that directory
    i. Using absolute pathnames
    ii. Using relative pathnames
6. Go back to your home directory
7. Create a directory `anynameyoulike` and change into it
8. Create a text file `yourfavoritename.txt` using an editor of your choice. Write something and save it.
9. Inspect your newly created file with `cat`, `more`, and `less`
10. Copy the file to `anothername.txt`
11. Edit the new file and save it.
12. Go back to your home directory

Text editors

- There are many text editors in linux: jedit, nedit, emacs, vi, pico, nano, kate.... It doesn't matter which one you use, but some are easier than others. For this lab class I recommend using gedit.

- File extensions – windows knows what program a file belongs to from its file extensions e.g. “sheet.xls” windows will know this is an excel file.

- Linux does not care what a file is called.

- Programs written in C are text files until compiled

Review of commands

- ksnapshot - takes a screen grab which can be incorporated into lab reports
- oowriter – Similar to MS word, lab reports can be written in this
- kate – text editor with syntax highlighting for C
- gnuplot – graph plotting tool
- ls - list directory content
- pwd - what directory am I in
- cp <from> <to> - copy
- mv <from> <to> - move (also acts as rename)
- rm – remove (remember there is no recycle bin!)

History and command line editing

- The shell usually remembers the last 500 commands you have entered.
- You can retrieve a previsouly entered command using the up-arrow-key.
- After a previously entered command has been retrieved (and before you hit RETURN) you can also edit the command line (using left/right arrows, backspace, delete, ...).

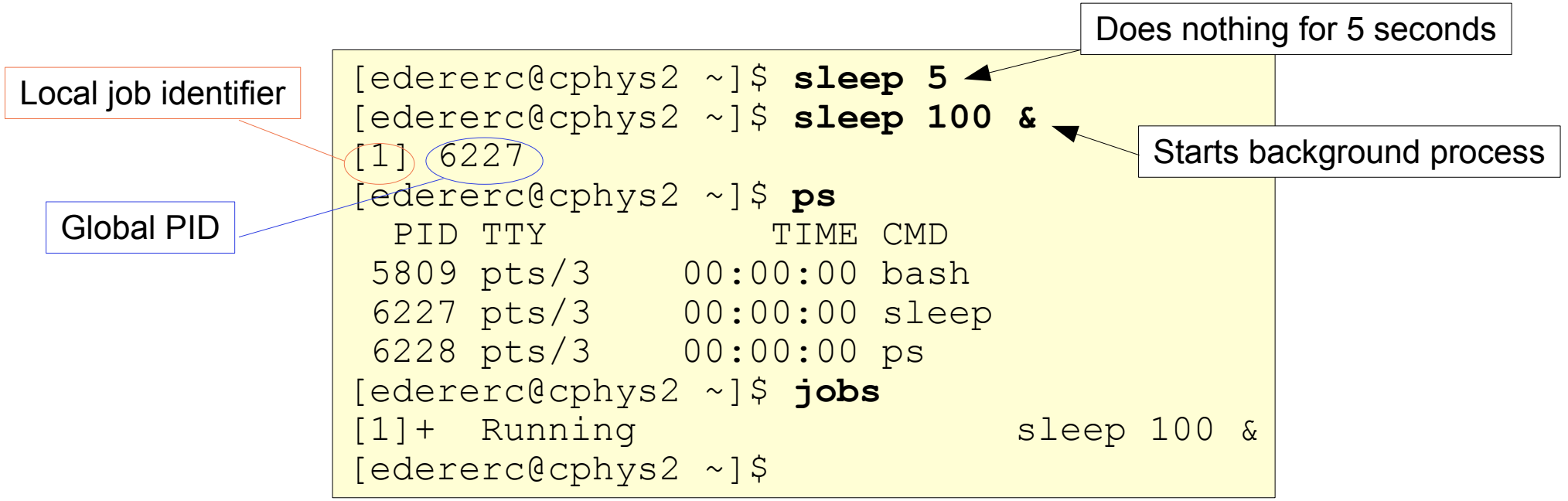
```
history    – shows command history list
!!         – refers to the last command
!n         – refers to command #n in the history list
!string    – refers to the last command starting with 'string'
```

- When you only type the first few letters of a command and hit the TAB key, the shell tries to complete your command. If there is no unique completion you can hit TAB twice to see the available options.

```
edererc> ls ~ed{TAB}
edererc> ls ~edererc/
edererc> ls ~edererc/D{TAB}
edererc> ls ~edererc/D{TAB}
edererc> ls ~edererc/D{TAB}{TAB}
Desktop/ Documents/
edererc> ls ~edererc/De{TAB}
edererc> ls ~edererc/Desktop/
```

Processes

- Every command that you enter starts at least 1 new process
- In addition, many processes related to various system functions are already running
- On a linux system many processes can run simultaneously
- Every process is uniquely identifed by a global process identification number: **PID**
- In addition, processes that you start in a terminal are assigned a (local) job identifier
- Processes can run in the **foreground** or in the **background**
- When you enter a command in a terminal the corresponding process usually runs in the foreground, i.e. It “blocks” the terminal
- To run a process in the background append the '`&`' symbol when entering the command.



Processes - continued

`gedit file.txt &`	– starts text editor as background process
`ps`	– displays information about selected processes (many command-line options)
`jobs`	– list all active jobs within current shell
`kill -[SIGNAL] PID`	– send signal to selected process
`STOP`	– suspend job execution, process is still active
`CONT`	– resume process execution
`TERM`	– terminate process
`KILL`	– abort process immediately
CTRL + C	– terminates current foreground process (IMPORTANT)
CTRL + Z	– suspends current foreground process
`bg` or `fg`	– resumes suspended process in the background/foreground

Input/Output redirection

3 “special files”:

- `stdin` - standard input, usually the keyboard
- `stdout` - standard output, usually the screen
- `stderr` - standard error output, usually the screen

`<`	– redirects `stdin`, e.g. `mail joe@somewhere.org < mail.txt`
`>`	– redirects `stdout`, e.g. `ls > filelist` (overwrites existing files!)
`>>`	– redirects `stdout` but appends to existing files
`|`	– uses output of left command as input for right command (input/output pipe) e.g. `ls /usr/bin | more`

Exercise 2:

1. Go into the directory `anynameyoulike` created during exercise 1
2. Print the contents of `yourfavoritename.txt` and then redirect its content to another file `ofyourchoice.txt`
3. Print the contents of `anothername.txt` and append it to `ofyourchoice.txt`
4. Inspect the content of `ofyourchoice.txt`
5. Create a process in the foreground `sleep 1000`
6. Suspend the foreground process
7. Resume the suspended process in the foreground
8. Terminate the process
9. Create a new process with the same command, then suspend it again
10. Resume it as a background process
11. Inspect all your active jobs
12. Kill the `sleep` process
13. Check whether the `sleep` process was really killed

Introduction to C programming

Where to get extra information

free c books

http://publications.gbdirect.co.uk/c_book/

http://www.oualline.com/style/

http://www.le.ac.uk/cc/tutorials/c/index.html

Compilers

- Programs are simply text files (referred to as source code)

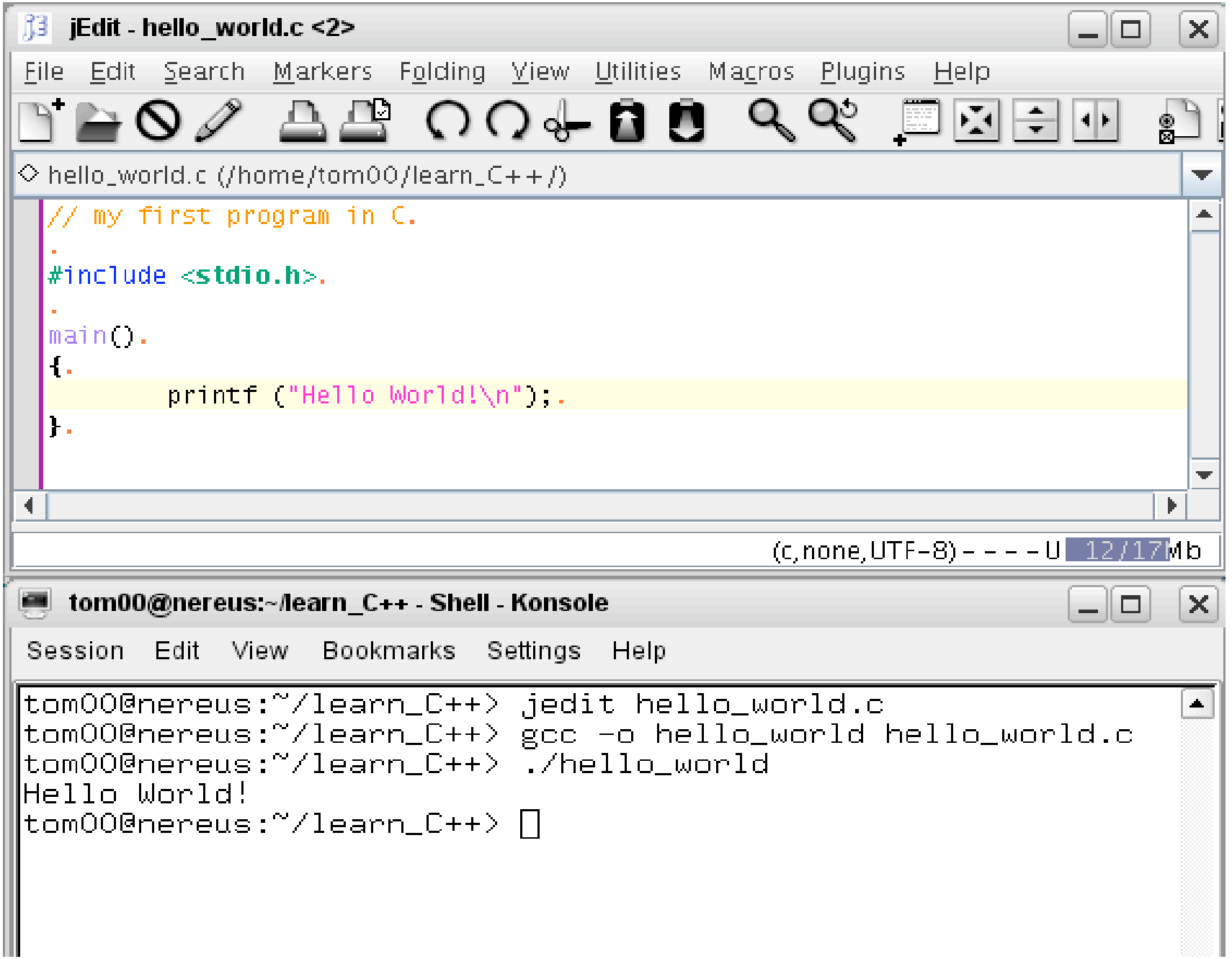
- file extension is usually .c e.g. myprog.c

- To convert to machine code so that the computer can run it the program needs to be compiled

- gcc -o myprog myprog.c

- here we input text file myprog.c, and the output file is called myprog

Simple program



Closer look at the simple program

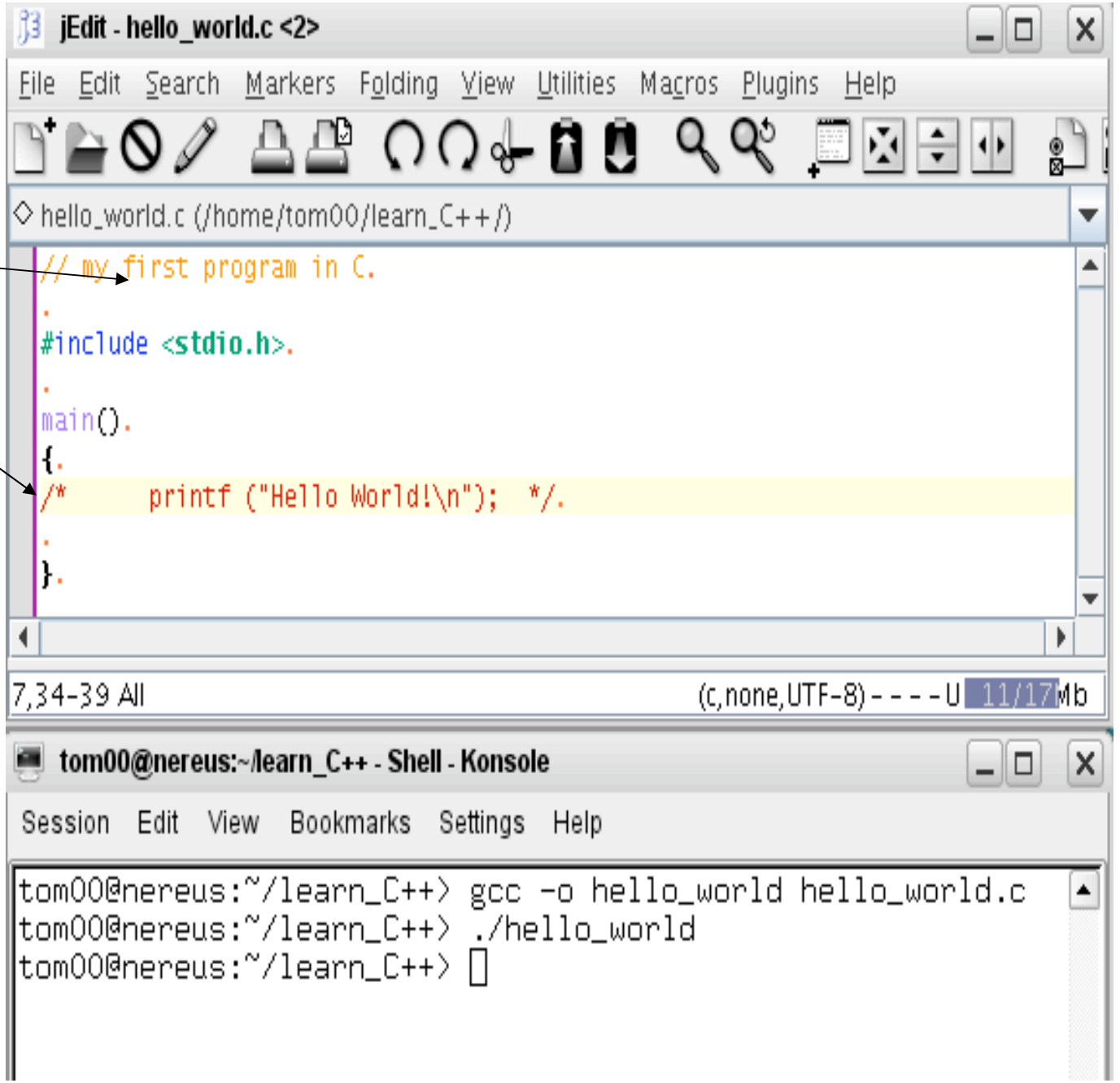
```
// my first program in C.
.
#include <stdio.h>.
main().
{.
        printf("Hello World!\n");.
}.
```

The function printf is contained in the library stdio.h

The program always starts with the function main. Other functions may also be included but main is the first one to be run.

Comments

Comments are ignored by the compiler, you should use comments to make notes as to what each section of the code does.

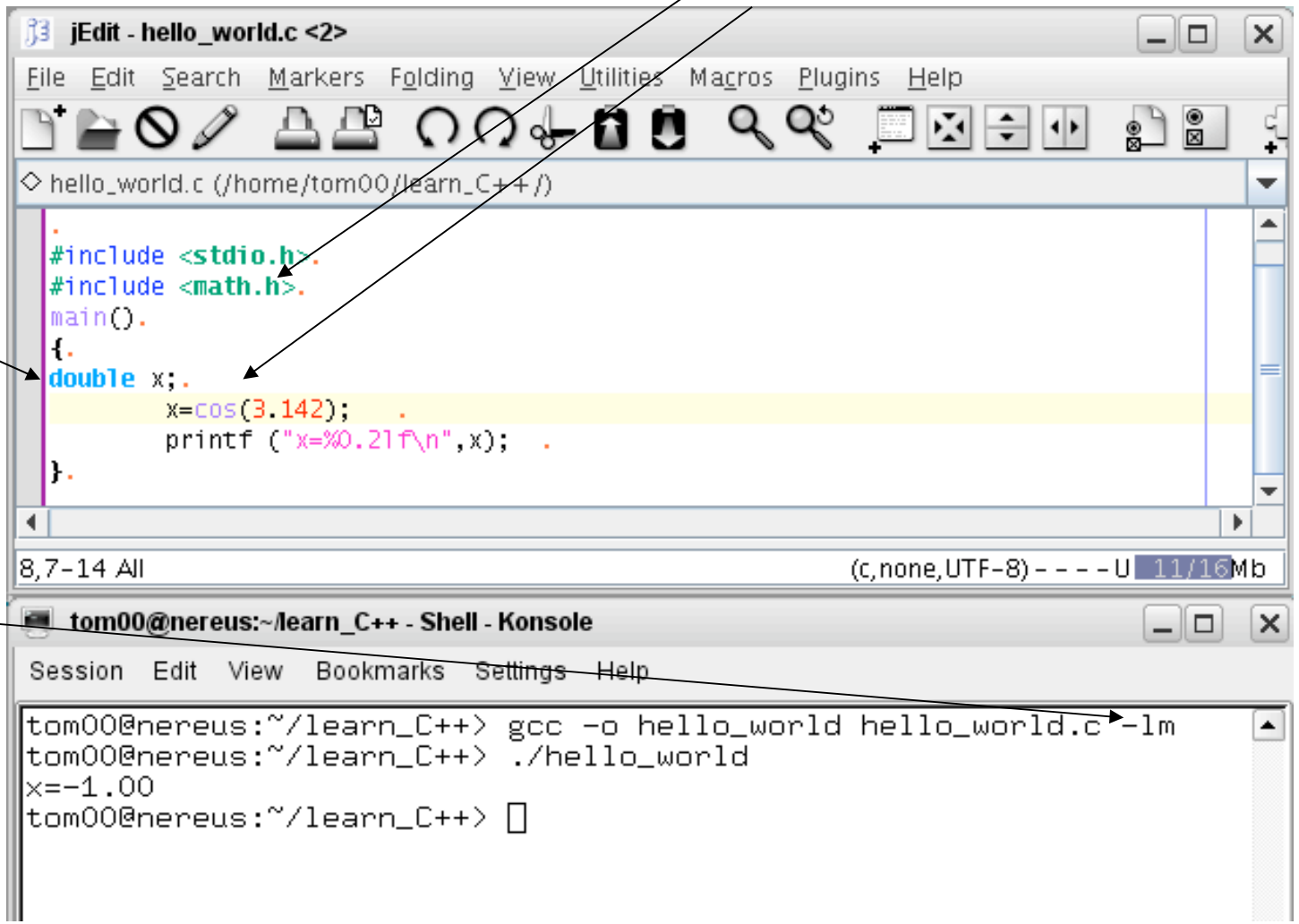


Introducing data types and maths library

The cos function is part of the library math.h

Every variable must fist be assigned so that the correct amount of memory is allocated.

The linker needs to be told where the library math.h can be found.



Variable types

type	use	size	range
char	character	8 bits	-128 to 127
unsigned char	character	8 bits	0 to 255
short	integer	16 bits	-32,768 to 32,767
unsigned short	integer	16 bits	0 to 65,535
int	integer	32 bits	-32,768 to 32,767
unsigned int	integer	32 bits	0 to 65,535
long	integer	32 bits	-2,147,483,648 to 2,147,483,647
unsigned long	integer	32 bits	0 to 4,294,967,295
float	real	32 bits	1.2E-38 to 3.4E+38
double	real	64 bits	2.2E-308 to 1.8E+308
long double	real	128 bits	3.4E-4932 to 1.2E+4932

data size depends on implementation but these are typical values

Input and output

```
%c  - characters
%d  - integers
%f  - floats
%lf - Long float
%s  - a string
\n  - new line
```

Print to screen:

printf("%s %d %f \n" , "red", 123, 0.02648);

Read in:

printf("What is your age: ");
scanf("%d ",&d);

```
#include <stdio.h>.
#include <math.h>.
.
main().
{  .
int d;.
        printf("what is your age: ");.
        scanf("%d",&d);.
        printf("you are %d %s\n",d,"years old");.
}.
.
```

Functions

As well as using pre built functions you may write your own

```
#include <stdio.h>.
#include <math.h>.
.
double func(double x);                    /*prototype*/.
.
main().
{.
        double z;.
        z=func(1.2347);.
        printf("z=%0.5f\n",z);.
}.
.
double func(double x)                     /*definition*/.
{.
        double y;.
        y=pow(sin(x),2)+pow(cos(x),2);.
        return y;.
}.
```

You must declare your functions, with the variables sizes they are going to use

function “func” returns the value of sin^2(x) +cos^2(x)

Control of flow with logical expressions

Operator	Operation
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	equal to
!=	not equal to

General form:

```
if( condition 1 )
   statement1;
else if( condition 2 )
   statement2;
else if( condition 3 )
   statement3;
else
   statement4;
```

```c
#include <stdio.h>.
#include <math.h>.
.
double func(double x);                    /*prototype*/.
.
main().
{.
        double z;.
        z=func(1.2347);.
        if(z<1) {.
        printf("z is less than 1\n");.
        }.
        else if(z>1){.
        printf("z is greater than 1\n");.
        }.
        else{.
        printf("z=1\n");.
        }.
}.
.
double func(double x)                              /*definition*/.
{.
        double y;.
        y=pow(sin(x),2)+pow(cos(x),2);.
        return y;.
}.
```

Control of flow with while statement

General form:

```
while(expression)
    statement
```

```
#include <stdio.h>.
#include <math.h>.
.
void func(double x);                    /*prototype*/.
.
main().
{  .
        double n;.
        n=0.0;.
        while(n<10){.
                func(n);.
                n=n+0.01;.
        }.
}.
.
.
/*given a double precision real number the .
functione returns if sin^2+cos^2 is 1*/.
        void func(double x){
        double z;.
        z=pow(sin(x),2)+pow(cos(x),2);.
        if(z<1) {.
        printf("z is less than 1\n");.
        }.
        else if(z>1){.
        printf("z is greater than 1\n");.
        }.
        else{.
        printf("z=1\n");.
        }.
}.
.
.
```

```
z=1
z=1
z=1
z=1
z=1
z is greater than 1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z is less than 1
z=1
z is greater than 1
z=1
z=1
z is less than 1
z=1
z=1
z=1
z=1
z=1
z is greater than 1
z=1
z=1
z=1
```

For loop

syntax:

for(initialization; condition for finishing; step)
statement

```
#include <stdio.h>.
#include <math.h>.
.
void func(double x);.
.
main(){  .
double n;.
        for(n=0; n<10; n=n+0.01).
        {.
                func(n);.
        }.
}.
.
void func(double x){.
        double z;.
        z=pow(sin(x),2)+pow(cos(x),2);.
        if(z<1){.
                printf("z is less than 1\n");.
        }.
        else if(z>1){.
                printf("z is greater than 1\n");.
        }.
        else{.
                printf("z=1\n");.
        }.
}.
```

Numerical error

```
#include <stdio.h>.
#include <math.h>.
.
void func(double x);                    /*prototype*/.
.
main().
{   .
        double n;.
        n=0.0;.
        while(n<10){.
                func(n);.
                n=n+0.01;.
        }.
}.
.
.
/*given a double precision real number the .
functione returns if sin^2+cos^2 is 1*/.
        void func(double x){
        double z;.
        z=pow(sin(x),2)+pow(cos(x),2);.
        if(z<0.99999999999)     {.
        printf("z is less than 1\n");.
        }.
        else if(z>1.000000001){.
        printf("z is greater than 1\n");.
        }.
        else{.
        printf("z=1\n");.
        }.
}.
.
```

```
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
z=1
```

Linking

- Once all the pieces of code are compiled the code is then gathered together (linked) into an executable.

- Linking allows functions that have already been written to be included into your executable.

- A set of prebuilt functions is called a library

- e.g. C comes with a maths library which contains functions like sin, cos, asin, tan, log, log10, ceil, floor ....

- We can build in the standard maths library as follows:

- gcc -o myprog myprog.c -lm

Compile and link