An Introduction to HPC and Scientific Computing

Lecture three: Introduction to Linux, compilers and build systems

Jan Novotny

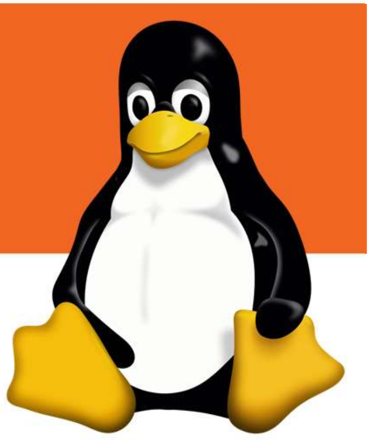
Oxford e-Research Centre,
Department of Engineering Science

Overview

In this **short** lecture we will learn about:

- Linux
- Compilers and make
- A little bit about arcus-htc – the machine that we will do the practicals on

Linux kernel



- 1991 Linus Torvalds:
  ``I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones …"
  ``It is NOT portable (uses 386 task switching etc), and it probably never will support anything other than AT-harddisks, as that's all I have :-("
- September 1991: first release of Linux kernel – 0.01
  ~10 000 lines of code
- Nowadays: Stable version 5.x.y
  – ~30 000 000 lines of code
- Personal computer, servers, embedded systems (wifi, routers, smart TVs, ...)

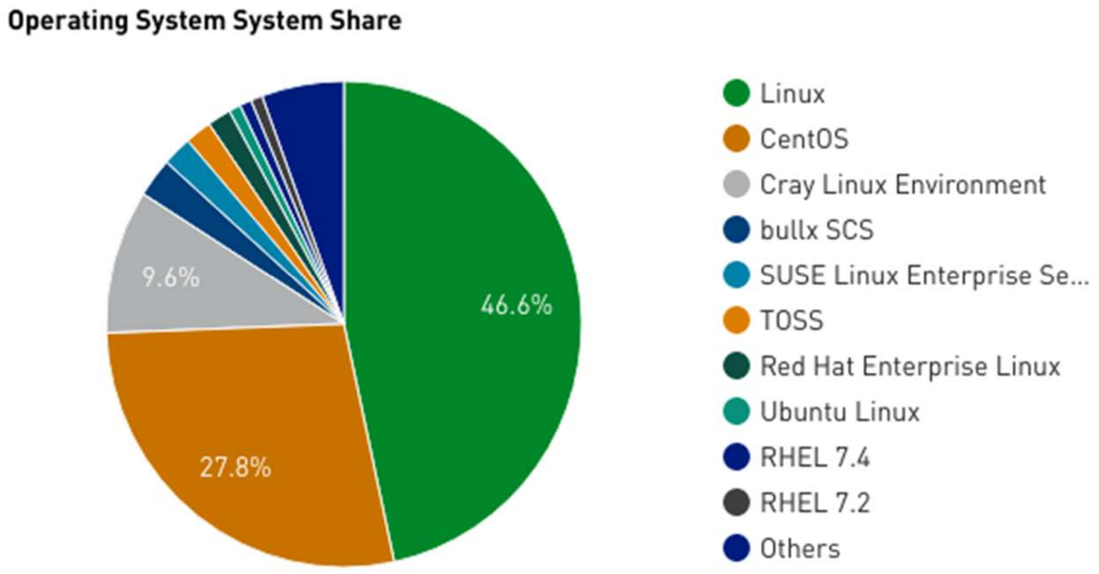


Image credits: Larry Ewing, top500.org

Linux

- Family of open-source Unix like OS based on Linux kernel
- Typically packaged in a Linux distribution: Linux kernel and supporting system software and libraries

Commercial
(Red Hat, SuSe)

No GUI, command line (CLI), or other (LAMPS)

Free and Open Souce
(Ubuntu, Fedora)

Windowing system
(X11, Wayland)

Desktop environment
(Gnome, KDE)

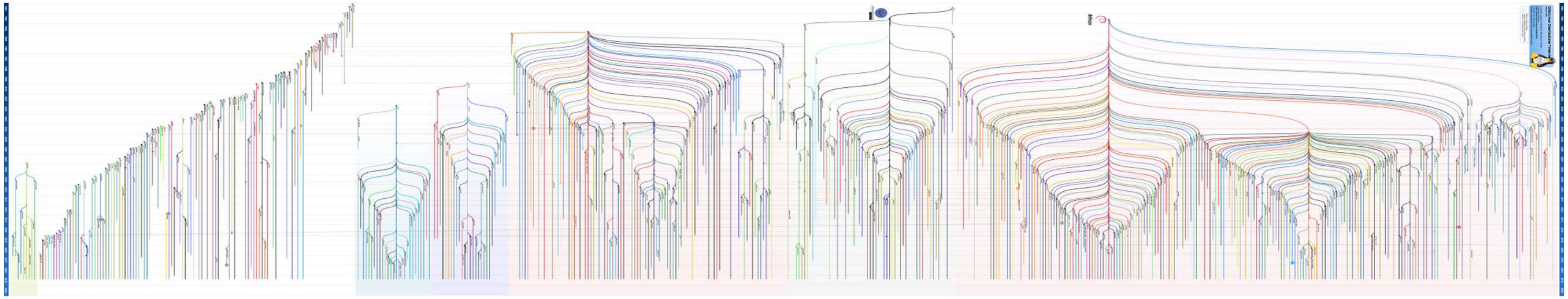


Image credits: Fabio Loli

Linux – basics

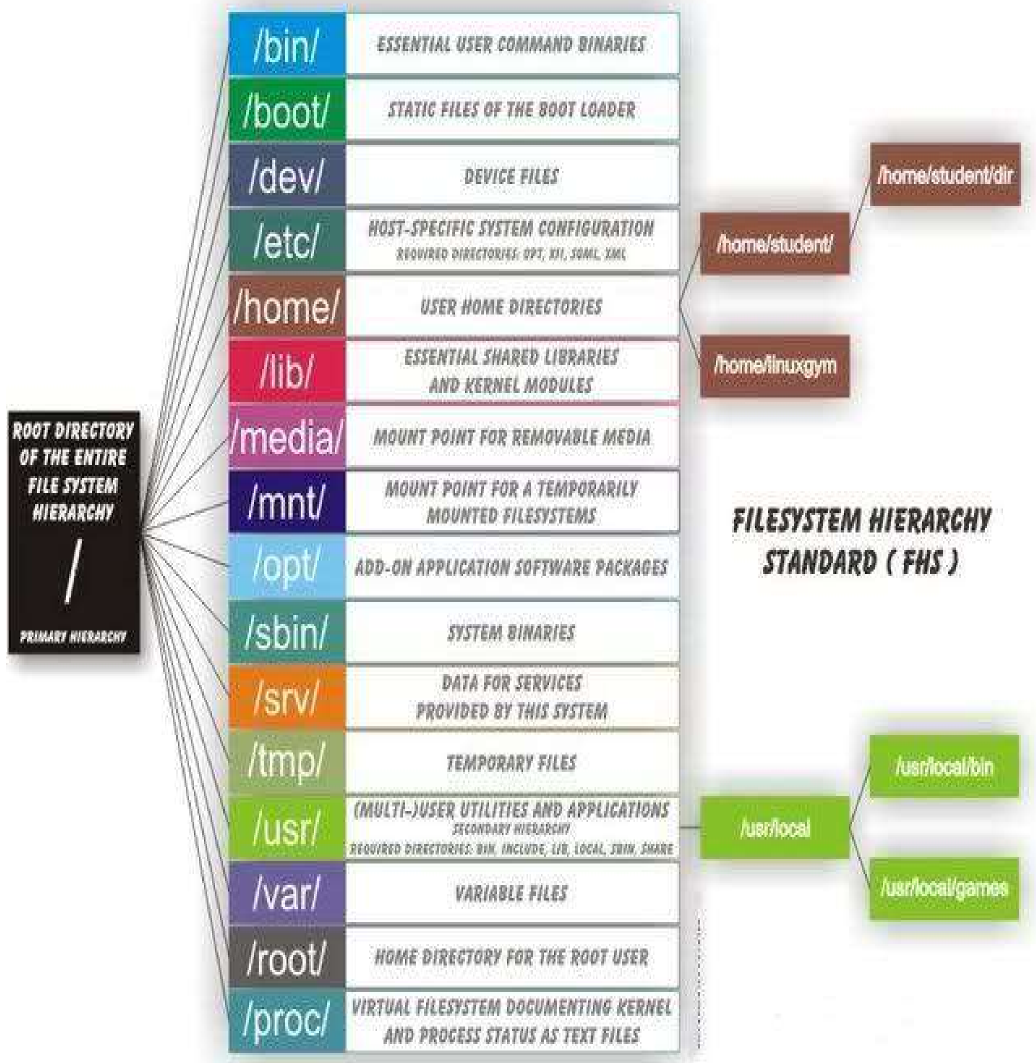

ROOT DIRECTORY OF THE ENTIRE FILE SYSTEM HIERARCHY
/
PRIMARY HIERARCHY
/bin/ ESSENTIAL USER COMMAND BINARIES
/boot/ STATIC FILES OF THE BOOT LOADER
/dev/ DEVICE FILES
/etc/ HOST-SPECIFIC SYSTEM CONFIGURATION REQUIRED DIRECTORIES: OPT, X11, SGML, XML
/home/ USER HOME DIRECTORIES
/lib/ ESSENTIAL SHARED LIBRARIES AND KERNEL MODULES
/media/ MOUNT POINT FOR REMOVABLE MEDIA
/mnt/ MOUNT POINT FOR A TEMPORARILY MOUNTED FILESYSTEMS
/opt/ ADD-ON APPLICATION SOFTWARE PACKAGES
/sbin/ SYSTEM BINARIES
/srv/ DATA FOR SERVICES PROVIDED BY THIS SYSTEM
/tmp/ TEMPORARY FILES
/usr/ (MULTI-)USER UTILITIES AND APPLICATIONS SECONDARY HIERARCHY REQUIRED DIRECTORIES: BIN, INCLUDE, LIB, LOCAL, SBIN, SHARE
/var/ VARIABLE FILES
/root/ HOME DIRECTORY FOR THE ROOT USER
/proc/ VIRTUAL FILESYSTEM DOCUMENTING KERNEL AND PROCESS STATUS AS TEXT FILES
/home/student/
/home/student/dir
/home/linuxgym
/usr/local
/usr/local/bin
/usr/local/games
FILESYSTEM HIERARCHY STANDARD (FHS)

Image credits: tecmint.com

- Linux system is build on a file system concept.
  - Stores not only OS, data files, program files, command files, software configuration files, special files (used to give access to system hardware, ...)

Four types of files:

- Directories
- Ordinary files
- Links
- Devices

Linux – basics

- Linux filenames: any characters, except '/' (256 length)

  * ? # & [spacebar]

- Case sensitive

  Abc.dat $\neq$ Abc.dat

- Permissions and ownerships

- Environment variables, package system

Linux – working in CLI

- Typically you see:

```
username@hostname:~$ ▌
```

- Keep track of the history of commands:
  - Up/down arrow, tab key, CTRL+R (backwards search)
- Usefull terminal shortcuts:
  - CTRL-c: end task
  - CTRL-u: clear line
  - CTRL-l: clear terminal
  - CTRL-a: move the cursor to the beginning of the current line
  - CTRL-e: move the cursor to the end of the current line
- Help:
  - help: help 'command' short info
  - 'command' -- help
  - man 'command'

Handling multiple files – wildcards

- Linux allows us to work with multiple files at once

- Pattern matching working:

  * : matches zero or more characters

  `*.*` Matches all files containing '.'

  ? : matches any single character at that position

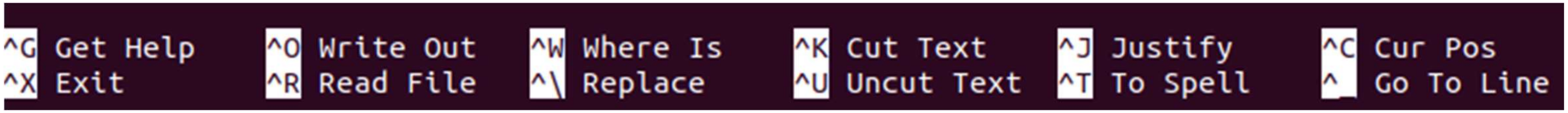
  `?ell?` Matches a 5 character long with 'ell' in the middle

  [] : will match any filename with characters that has one of those characters in that position

  `[1-9].txt` Matches any filename with single numeric character ending with .txt

Editor

- We will use: nano
  - CTRL-G : brings help CTRL-S : save
  - CTRL-X : exit
- Note: pressing **ctrl+g** no ctrl+shift+g



```
^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos
^X Exit        ^R Read File   ^\ Replace     ^U Uncut Text  ^T To Spell    ^_ Go To Line
```

- How to quit vim or emacs?

Vim/vi -- ":q!" -- quit without save

Emacs -- "ctrl-x, ctrl-c" -- if file modified will ask to quit without save save

Computing – You only learn by doing it

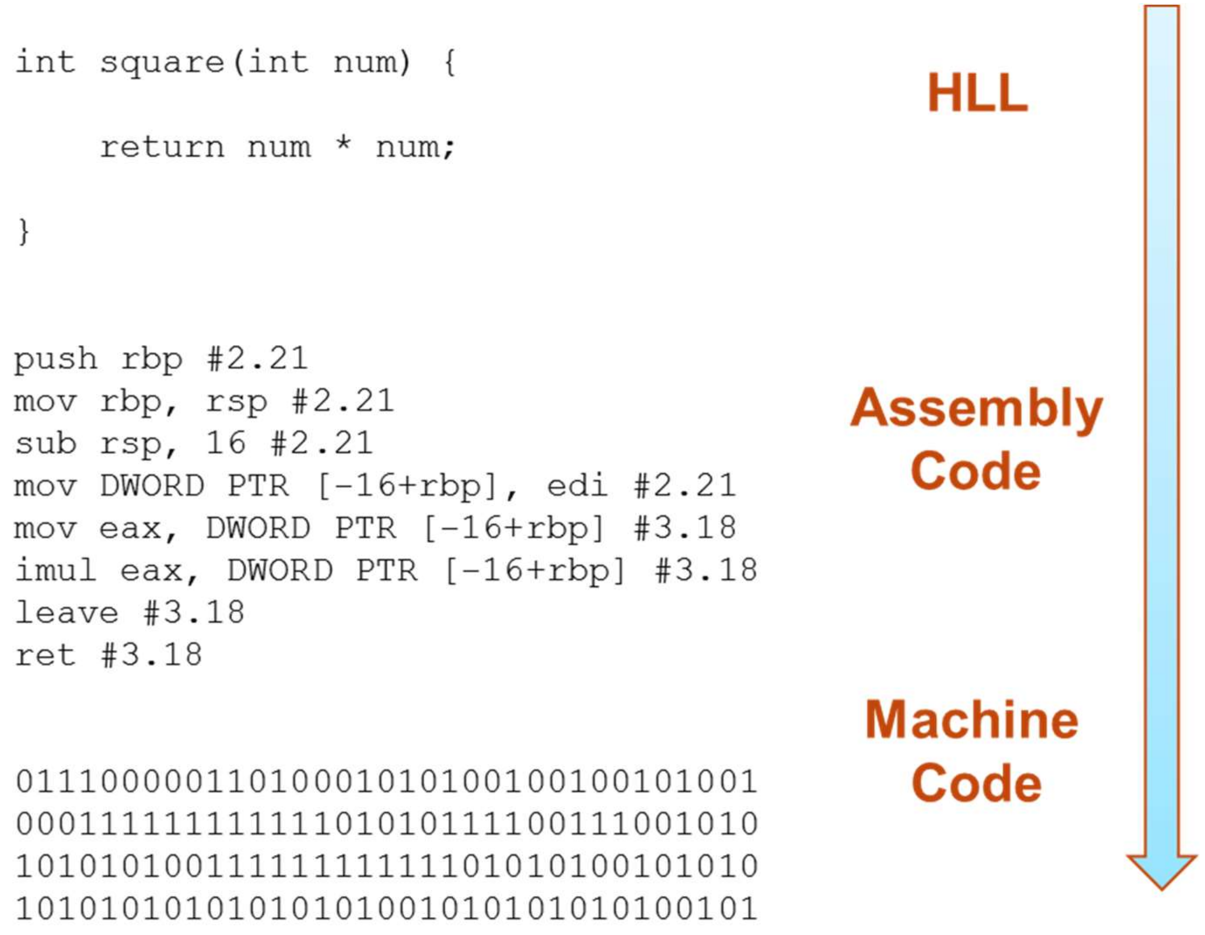
- In Unix you have a large number of relatively simple tools which can easily be chained together to do something quite complicated.
- Under Windows you tend to have one big tool that does complicated things. Great and easy to use if you want to do what it can do but not great once you stray outside those boundaries.

Anyway, we’ll learn Linux by doing it!

Using Compilers

- As you learnt earlier a compiler converts a High-level language such as C into the machine code that the computer understands

- Let’s look a little more at that and how it works under Linux



```
int square(int num) {

    return num * num;

}
```

```
push rbp #2.21
mov rbp, rsp #2.21
sub rsp, 16 #2.21
mov DWORD PTR [-16+rbp], edi #2.21
mov eax, DWORD PTR [-16+rbp] #3.18
imul eax, DWORD PTR [-16+rbp] #3.18
leave #3.18
ret #3.18
```

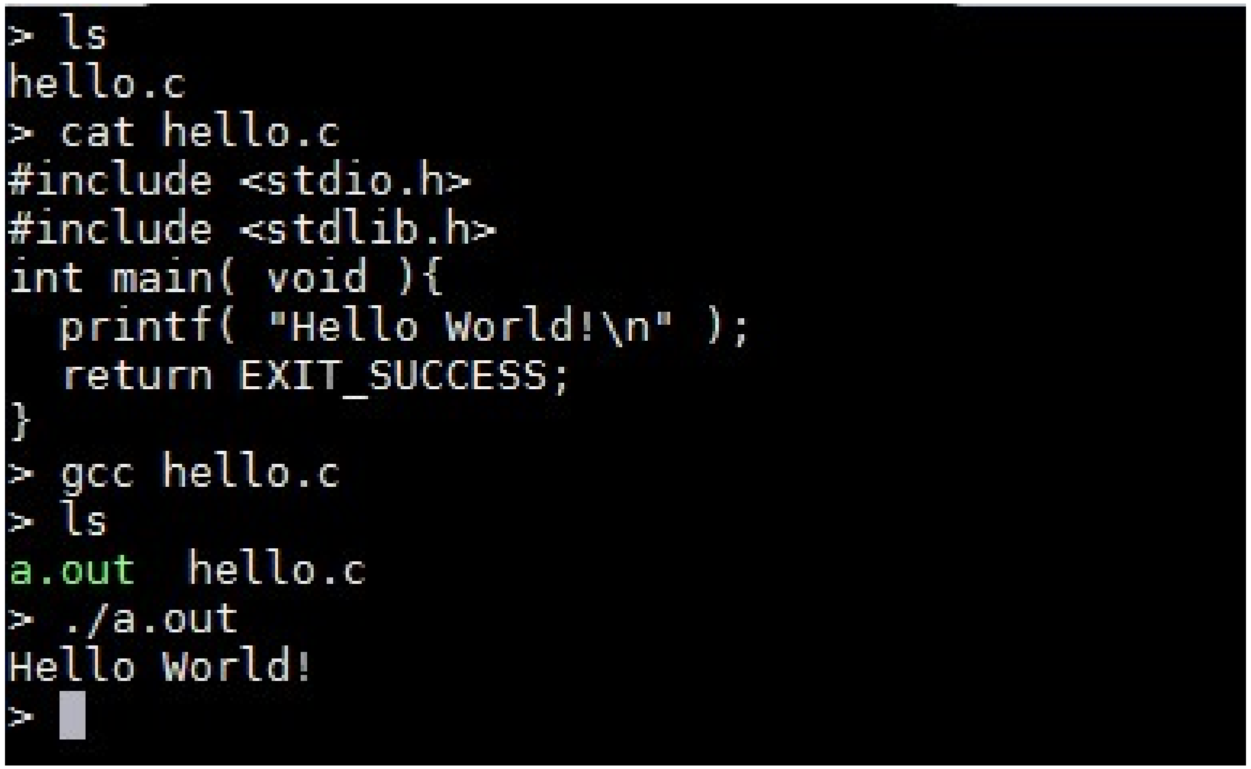
```
01110000011010001010100100100101001
00011111111111101010111100111001010
10101010011111111111101010100101010
10101010101010101001010101010100101
```

https://godbolt.org/

There is No One True Compiler

- There are many compilers available under Linux
- The most commonly used are
  - gcc – The GNU Compiler
    - Truly free, comes with every Linux
  - icc – The Intel Compiler
    - Commercial (i.e. costs money), not always available, but generally produces executables that run faster than those from the GNU compiler

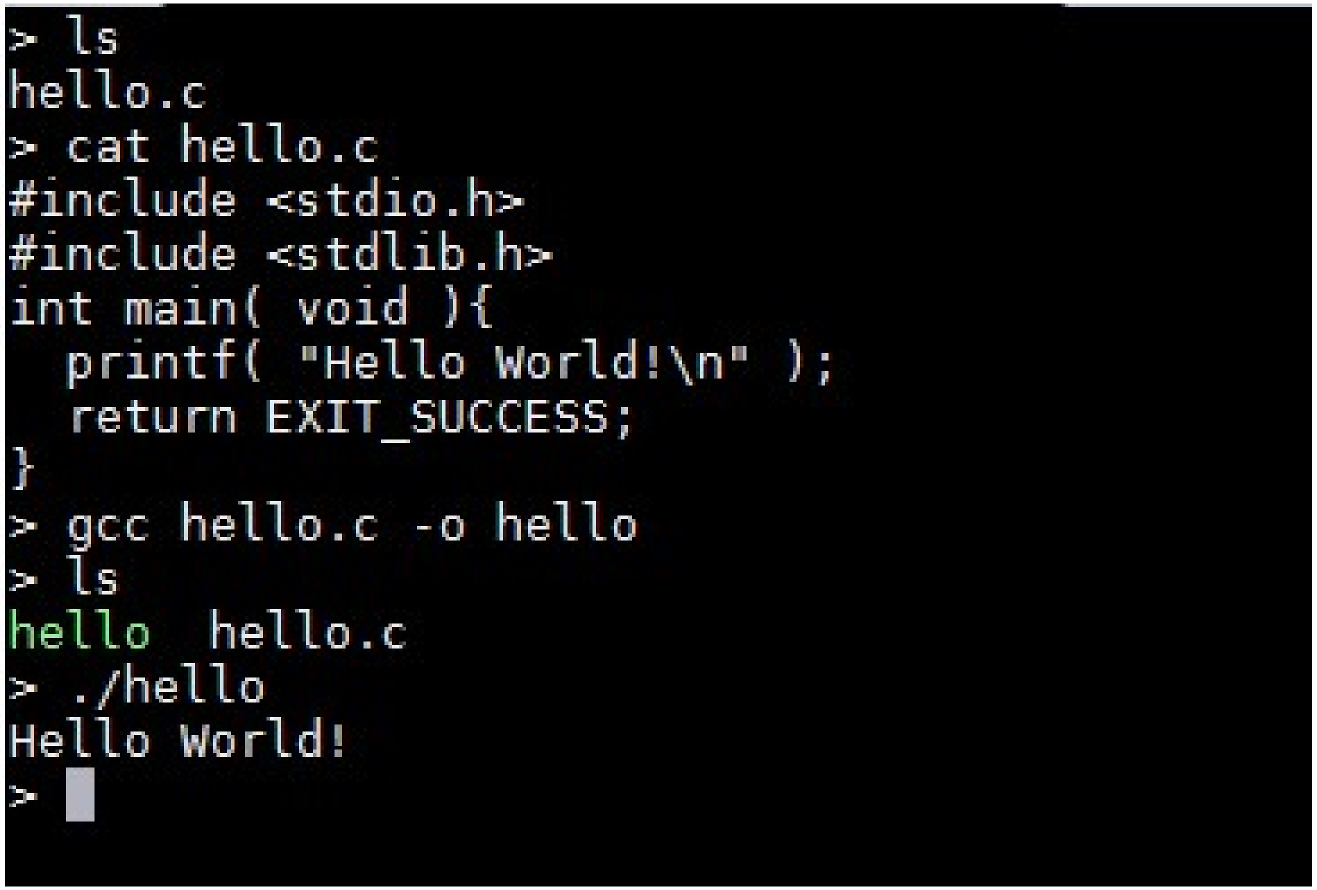
Using A Compiler Under Linux



```
> ls
hello.c
> cat hello.c
#include <stdio.h>
#include <stdlib.h>
int main( void ){
  printf( "Hello World!\n" );
  return EXIT_SUCCESS;
}
> gcc hello.c
> ls
a.out  hello.c
> ./a.out
Hello World!
>
```

- Note
  - Type of input/output: stdin, stdout, stderr
  - You should always end your C file names with `.c` – confusion will occur if you do not!
  - The default name for an executable under Linux is a.out
  - You run an executable by using its name: **./a.out**

Compilers with Flags

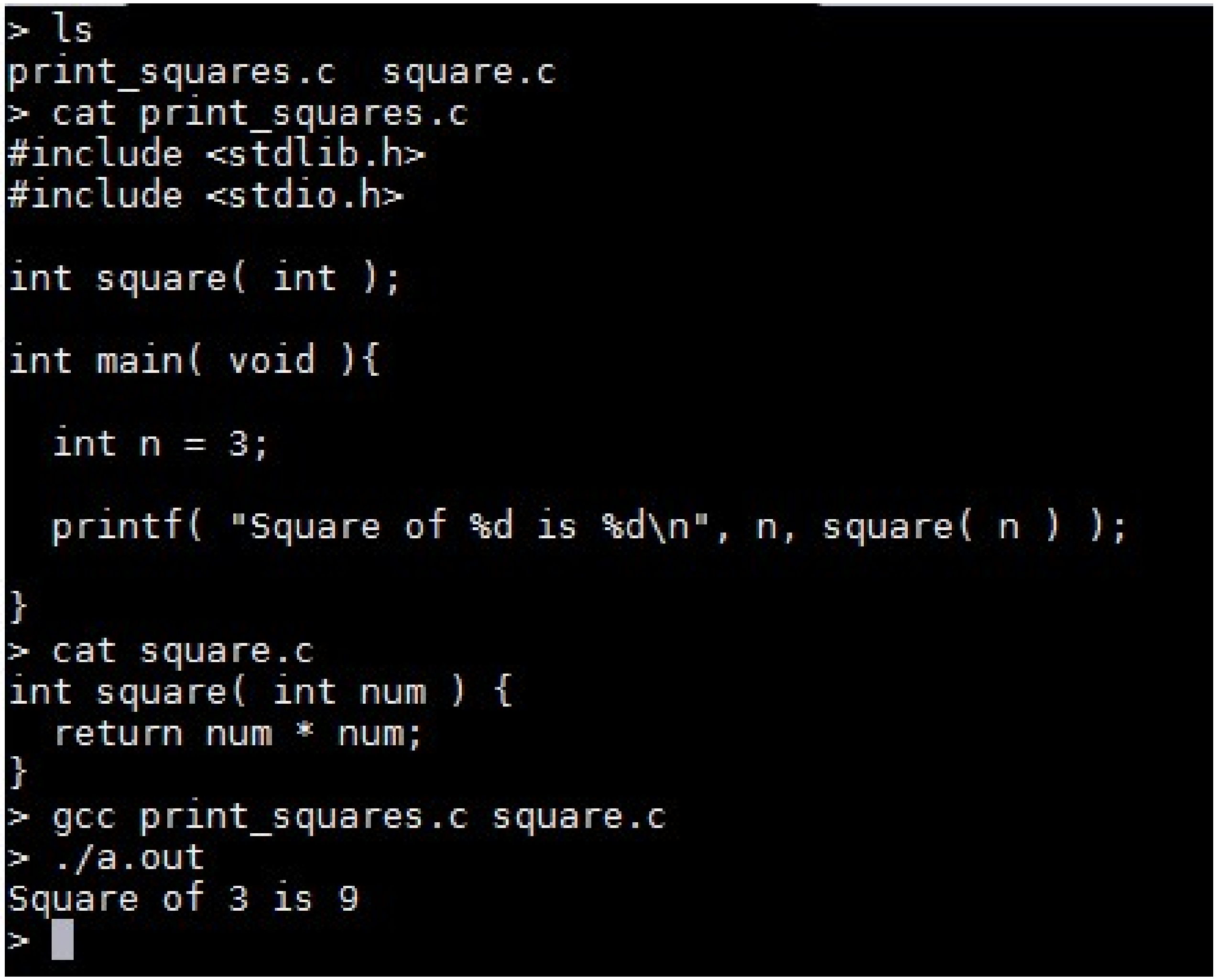


```
> ls
hello.c
> cat hello.c
#include <stdio.h>
#include <stdlib.h>
int main( void ){
  printf( "Hello World!\n" );
  return EXIT_SUCCESS;
}
> gcc hello.c -o hello
> ls
hello  hello.c
> ./hello
Hello World!
>
```

- Compilers can take many flags which alter how they behave
  - Here we make the executable have a more sensible name
  - In the practical we will see more examples of this

Programs in Multiple Files

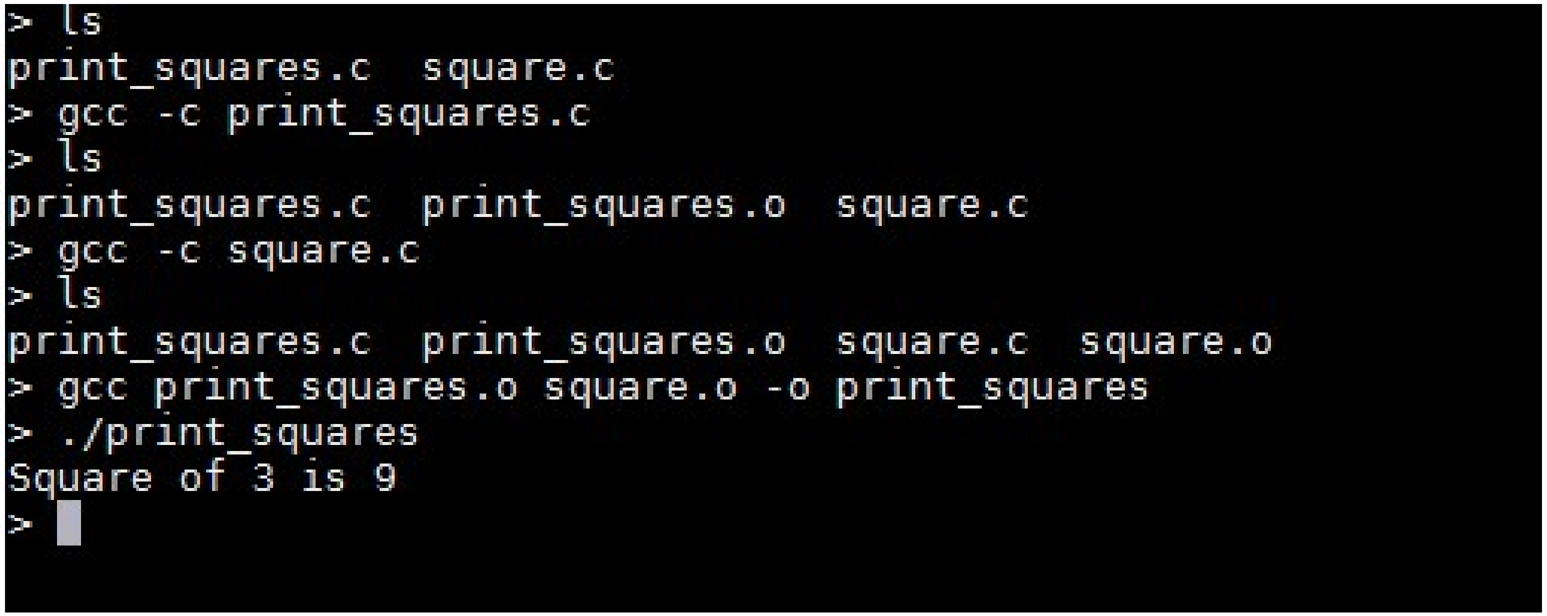
- Note a program need not be all in one file:



```
> ls
print_squares.c  square.c
> cat print_squares.c
#include <stdlib.h>
#include <stdio.h>

int square( int );

int main( void ){

  int n = 3;

  printf( "Square of %d is %d\n", n, square( n ) );

}
> cat square.c
int square( int num ) {
  return num * num;
}
> gcc print_squares.c square.c
> ./a.out
Square of 3 is 9
> 
```

Compiling and Linking

- Note there are actually two phases in this process
- We first compile each file to a corresponding *object file*
  - Thus each source file has a corresponding object file
- Then all the object files are *linked* together to produce the executable
- By default both are done
- The `-c` flag can be used to force only the compilation stage



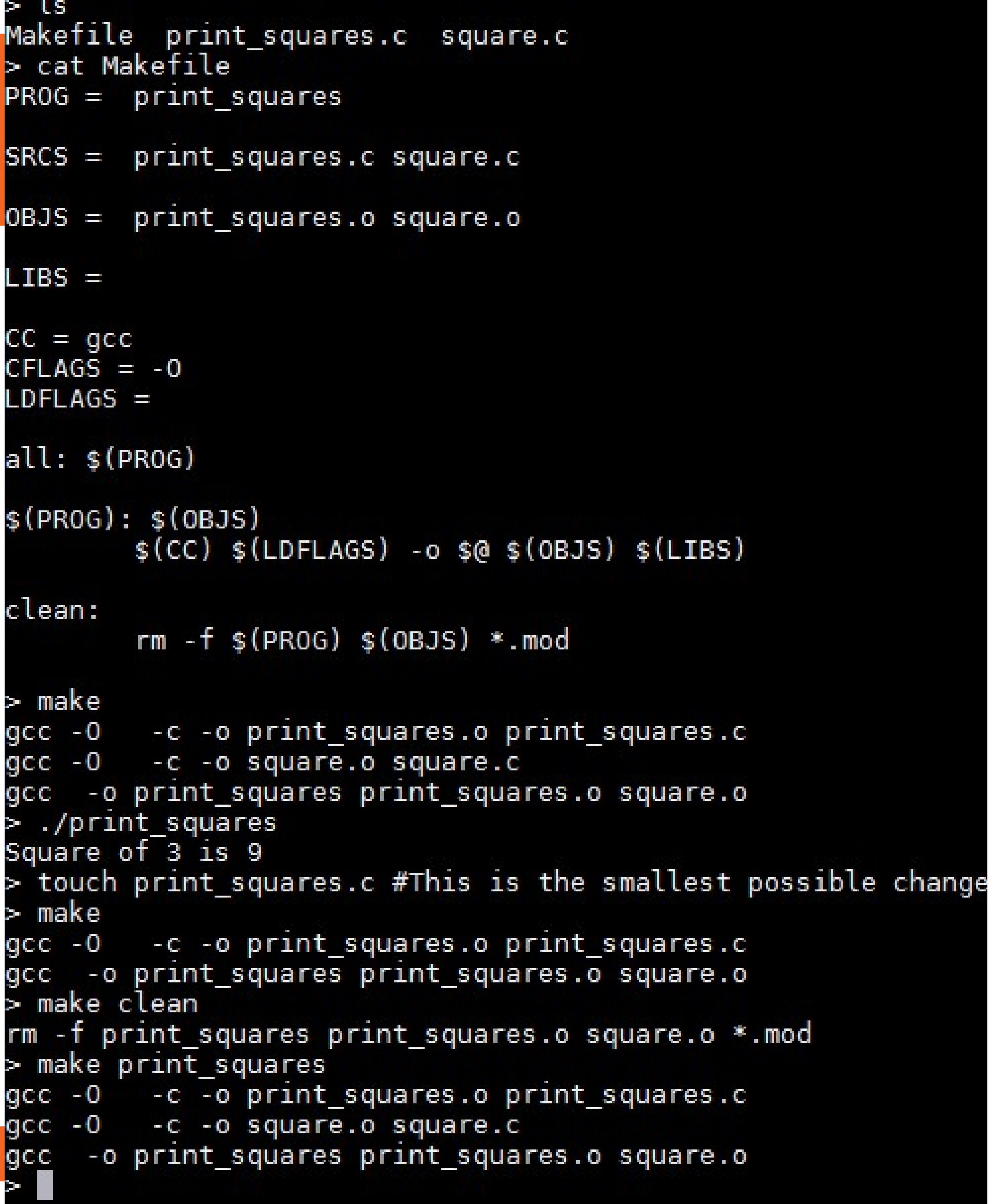
```
> ls
print_squares.c  square.c
> gcc -c print_squares.c
> ls
print_squares.c  print_squares.o  square.c
> gcc -c square.c
> ls
print_squares.c  print_squares.o  square.c  square.o
> gcc print_squares.o square.o -o print_squares
> ./print_squares
Square of 3 is 9
> 
```

Compiling and Linking

- Why is this useful?
- Let’s pretend we have a program in 100 different files
- We now compile all those 100 files and link the resulting object files together to produce an executable
- We now change just one of the source files
- Without the 2 stage system we would have to recompile everything
- But now we can just compile the one source file that has changed to a new object file, and then relink that with the existing 99 old ones to produce a new executable
- This can save a lot of time in a big program!
- It is so useful there is a special utility called `make` that takes advantage of this
  - We don’t need to know much about make as we will provide the makefiles for all the exercises
  - We mainly need to know how to use it

Using make

- Note how the Makefile contains a set of rules that `make` interprets
  - Don’t worry, we will always provide this!
- Note how it also automatically works out what files need to be compiled and only compiles them
- Also `make clean` is very common – clean up and leave the files as they were before any compilation occurred



```
> ls
Makefile  print_squares.c  square.c
> cat Makefile
PROG =  print_squares

SRCS =  print_squares.c square.c

OBJS =  print_squares.o square.o

LIBS =

CC = gcc
CFLAGS = -O
LDFLAGS =

all: $(PROG)

$(PROG): $(OBJS)
        $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LIBS)

clean:
        rm -f $(PROG) $(OBJS) *.mod

> make
gcc -O   -c -o print_squares.o print_squares.c
gcc -O   -c -o square.o square.c
gcc  -o print_squares print_squares.o square.o
> ./print_squares
Square of 3 is 9
> touch print_squares.c #This is the smallest possible change
> make
gcc -O   -c -o print_squares.o print_squares.c
gcc  -o print_squares print_squares.o square.o
> make clean
rm -f print_squares print_squares.o square.o *.mod
> make print_squares
gcc -O   -c -o print_squares.o print_squares.c
gcc -O   -c -o square.o square.c
gcc  -o print_squares print_squares.o square.o
>
```

Arcus-htc

Advanced Research Computing (ARC) is a central resource available to any Oxford University researcher who needs high performance computing (HPC) from any Division or Department. ARC is an IT Services facility operated in partnership with the Oxford e-Research Centre.

The ARC High Throughput Cluster (or Arcus-HTC) has been created by repurposing the original Arcus cluster (Arcus-A).

Arcus-htc – login

- Cluster visible through the main GATEWAY:

```
ssh -CX username@oscgate.arc.ox.ac.uk
```

- From oscgate we can log to arcus-htc:

```
ssh -CX username@arcus-htc
```

- Now we are on the cluster were a job scheduler is working. To get to access to a free node we need to use the SLURM command:

```
srun --gres=gpu:1 --pty /bin/bash
```

Arcus-htc – software environment

- You gain access to software via the module system
  - `module load` gives access to the software
  - `module avail` lists what is available
  - `module list` lists what is currently loaded
  - `module purge` removes everything – often useful to start from a blank sheet to avoid confusion

What have we learnt?

- Linux is the OS most supercomputers use
- You interact with it via the command line
- There are many compilers, we will use the GNU and Intel ones
- Compilation is really two phases, compilation to an object file, and then linking the object files together to produce the executable
- The make utility can take advantage of this to speed up compilation, especially for large projects