CSC 7700: Scientific Computing

Module A: Basic Skills

Lecture 1: Preliminaries

Dr Frank Löffler / Dr Allen

August 26 2010

Unix / Linux

Unix History

- Originally developed in 1969 by a group of AT&T employees at Bell Labs
- Today's Unix systems split into various branches
- Adoption of Unix by commercial startups, most notably Solaris, HP-UX and AIX
- In contrast to: Unix-like operating systems such as Linux and BSD descendants
- Designed to be portable, multi-tasking and multi-user in a time-sharing configuration

Unix Concepts

- Characterized by various concepts:
  - use of plain text for storing data
  - a hierarchical file system
  - treating devices and some forms of inter-process communication as files
  - use of a large number of software tools that can be strung together, as opposed to using a single monolithic program that includes all of the same functionality
- "Operating system" consists of many utilities along with the kernel
- Common executable and Linkable Format: ELF
- Filesystem Hierarchy Standard for common file system layout

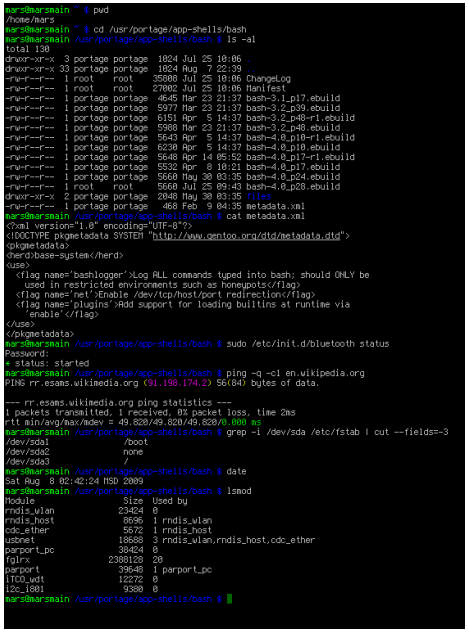
Linux

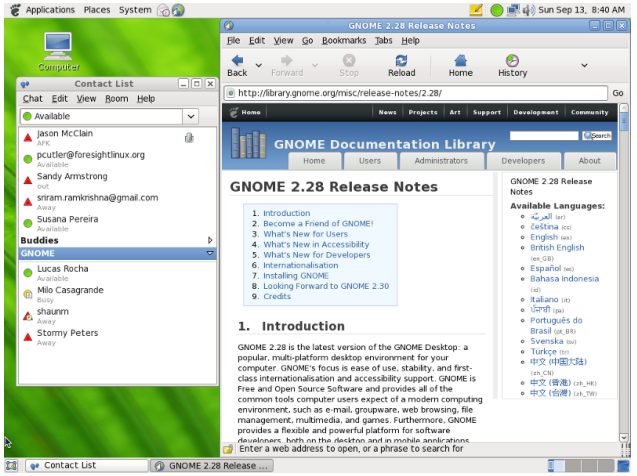
- Unix-like computer operating system using the Linux kernel
- Predominantly known for its use in servers
- Free and open source software collaboration
- Typically packaged in a format known as a Linux distribution, e.g. Fedora, Debian, Ubuntu, openSuse
- Include the Linux kernel and all of the supporting software required to run a complete system
- Main supporting user space system tools and libraries from the GNU Project
- Commonly used software on desktop systems: Firefox, OpenOffice, Gimp, Inkscape

Shells

Shells

- Provides an interface for users of an operating system
- Originates from shells being an outer layer of interface between the user and the internals of the operating system (the kernel)
- Two categories: command-line and graphical



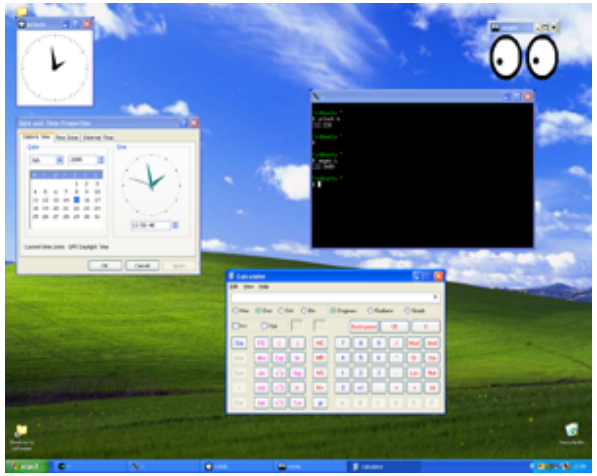


CLI Shells

- Often simply called "shells"
- Provide a command-line interface (CLI)
- Mechanism for interacting with a computer operating system or software by typing commands to perform specific tasks
- Text-only interface
- Command-line interpreter receives, analyses, and executes requested commands, examples: sh, ksh, csh, tcsh, bash
- Can be embedded in GUIs
- Most popular: BASH

GUIs in Unix/Linux

- Using X Window System
  - Provides windowing on computer displays and manages keyboard and mouse control functions
  - Does not mandate the user interface (task of Window managers)
  - Specifically designed to be used over network connections
  - Can be tunneled, e.g. via Secure Shell
  - Lots of implementations, e.g. XFree86, X.Org, DECwindows, MacX, Cygwin/X, Exceed
- Window managers handle design of interface, e.g. Gnome, KDE



Basic commands

Function	Command
Show current directory	`pwd`
List directory content	`ls [file(s) / directorie(s)]`
Change directory	`cd [directory name]`
Create directory	`mkdir <directory name(s)>`
Remove directory	`rmdir <directory name(s)>`
Remove files	`rm <file name(s)>`
Rename/move file or directory	`mv <source> <destination>`
Logout	`logout` (typically `Crtl+d`)

Path essentials

Directory separator	`/`
Current directory	`./`
Parent directory	`../`
Home directory (*)	`~/`
Previous directory (*)	`-` (to be used as `cd -`)
Wildcard 'any character'(*)	`?`
Wildcard 'any number of characters' (*)	`*`
Configuration files	`~/.?*`
Allowed characters in filename	anything but `\0` and `/`

(*) Shell-specific

Special shell characters

Character	Meaning
#	Comment until end of line
;	Command separator
"	Partial quoting
'	Full quoting
\	Escaping
*	Wildcard 'any number of characters'
?	Wildcard 'any character'
$	Variable substitution

Secure Shell

Secure Shell

- Short: SSH
- Network protocol, using a secure channel
- Uses public-key cryptography to authenticate users
- Uses the client-server model
- Some uses
  - login to a shell
  - secure file transfer
  - network port tunneling and forwarding
  - virtual private networks
  - mounting remote directories
- Implementations: OpenSSH, PuTTY, ...

Secure Shell Usage

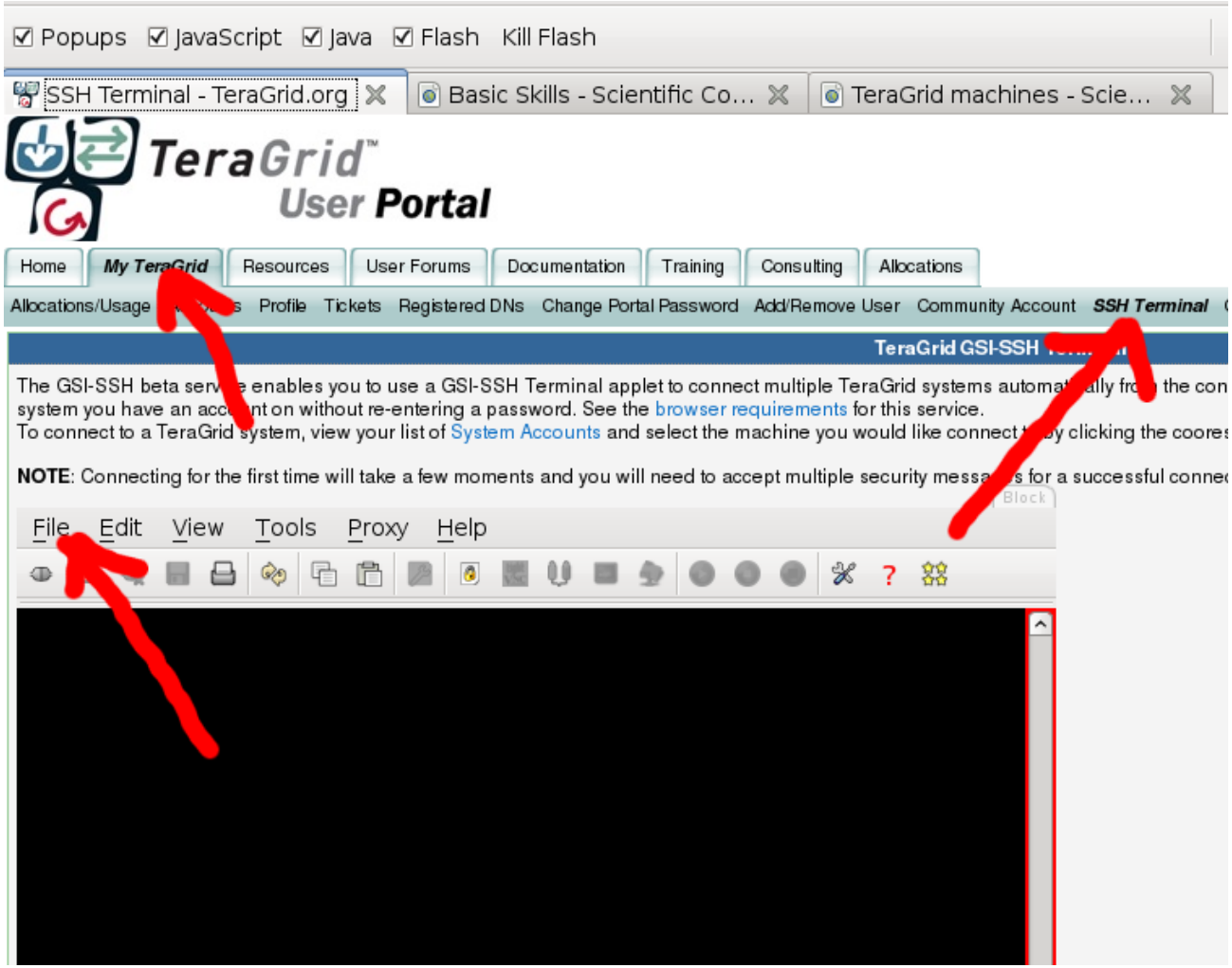
- OpenSSH
  - Command line interface
  - Free and Open Software
  - Installed on almost all Linux machines

Command	Effect
`ssh [user@]<host>`	Login to shell at `host`
`ssh [user@]<host> command`	Execute `command` at `host`
`ssh -Y <host>`	Enable X-Forwarding to `host`

- PuTTY
  - Originally written for Windows
  - Free and Open Software
  - Self-contained executable, requires no installation
  - Manual available online: http://the.earth.li/ sgtatham/putty/0.60/htmldoc/

TeraGrid Portal - GSISSH

https://portal.teragrid.org/



GSISSH

Text Editors

Text Editors

- Program used for editing plain text files (as apposed to e.g. Word documents)
- Examples: Vi, Emacs, nano, Notepad, TextEdit
- Typical features
  - String searching algorithm
  - Cut, copy, and paste
  - Text formatting
  - Undo and redo
  - Syntax highlighting
- Specialized Editors for e.g. Source code, IDEs, HTML, TeX

Plain Text Files

- Structured as a sequence of lines
- End of a text file is sometimes denoted by a special characters (EOF), systemdependend
- End of line is indicated by EOL marker
  - `LF`: Unix, Linux, Max OS X, BeOS, Amiga
  - `CR+LF`: DOS, Windows, OS/2, Symbian
  - `CR`: MacOS up to version 9
- Character encodings, e.g.:
  - ASCII: simple, but very limited
  - ISO 8859-?: Extensions of ASCII, but still very limited
  - UTF-8: backwards compatible to ASCII, but very large character set
- EOL and encoding conversions sometimes neccessary

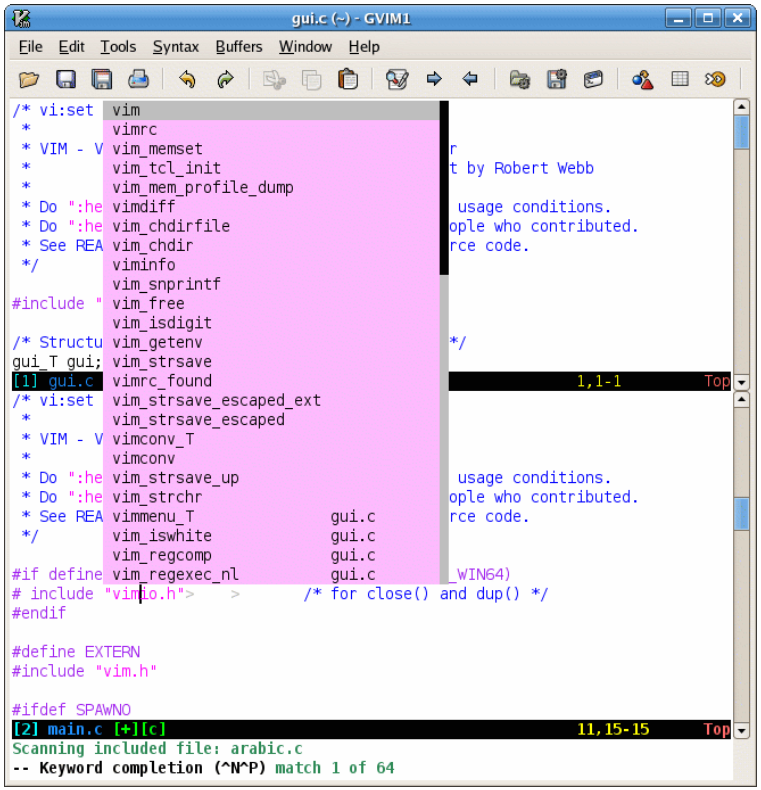
Text Editor Examples: Nano

- Using text terminal (curses-based)
- Free replacement for earlier editor 'pico' (included in 'pine')
- Very easy, but also very basic

```
  GNU nano 2.1.2-svn      File: ./Download/SVN/nano/src/nano.c

/* Disable mouse support. */
void disable_mouse_support(void)
{
    mousemask(0, NULL);
    mouseinterval(oldinterval);
}

/* Enable mouse support. */
void enable_mouse_support(void)
{
    mousemask(ALL_MOUSE_EVENTS, NULL);
    oldinterval = mouseinterval(50);
}

/* Initialize mouse support.  Enable it if the USE_MOUSE flag is set,
 * and disable it otherwise. */
void mouse_init(void)
{
    if (ISSET(USE_MOUSE))

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

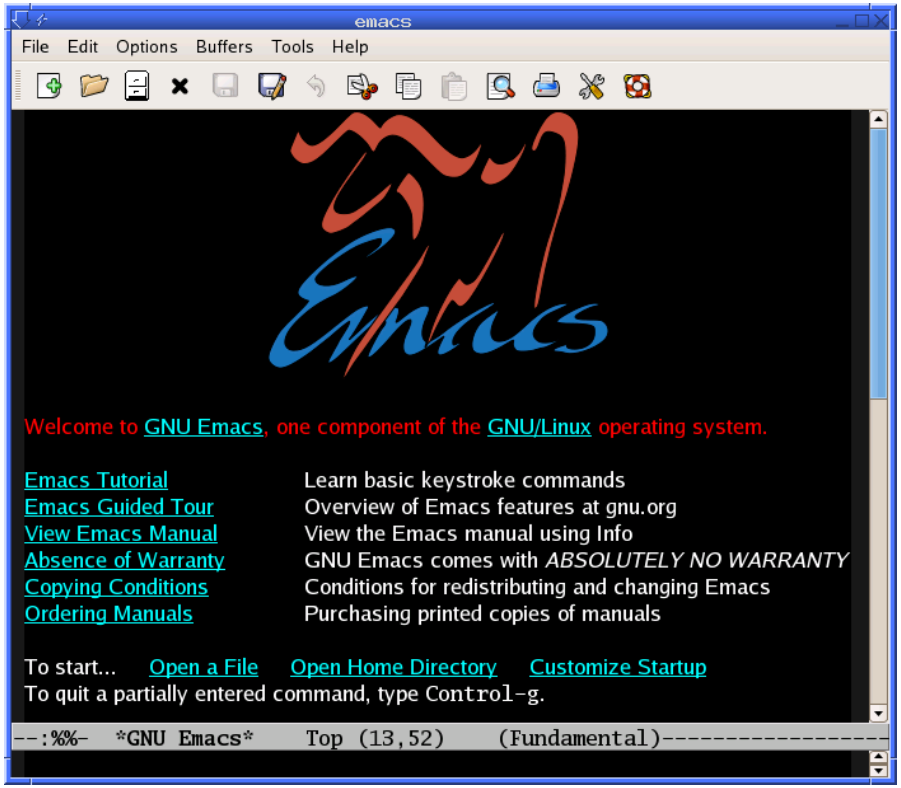
Text Editor Examples: Vim

- Modal editing (insert-mode and command mode) by switching entire keybard in and out of modes
- Can (but does not have to be) used entirely with the keyboard
- Minimal use of Meta keys
- Can be extensively customized
- Available for virtually every operating system



Text Editor Examples: Emacs

- Very feature-rich
- Highly customizable
- Extensive use of Meta keys
- Available for virtually every operating system



Text Editor Recommendations

- No single recommendation (There isn't **the** tex editor.)
- Choose whatever **you** like
- Important aspects: Use editor that can
  - handle UTF-8 encoding (and use it)
  - understand end respect different EOL styles
- Nice-to-have aspects
  - Syntax-highlighting
  - Available on multiple OSs
- All three recommendations (nano, vim, emacs) fulfill above points, but many others do as well
- Again: choose whatever **you** like

Compiling and Linking

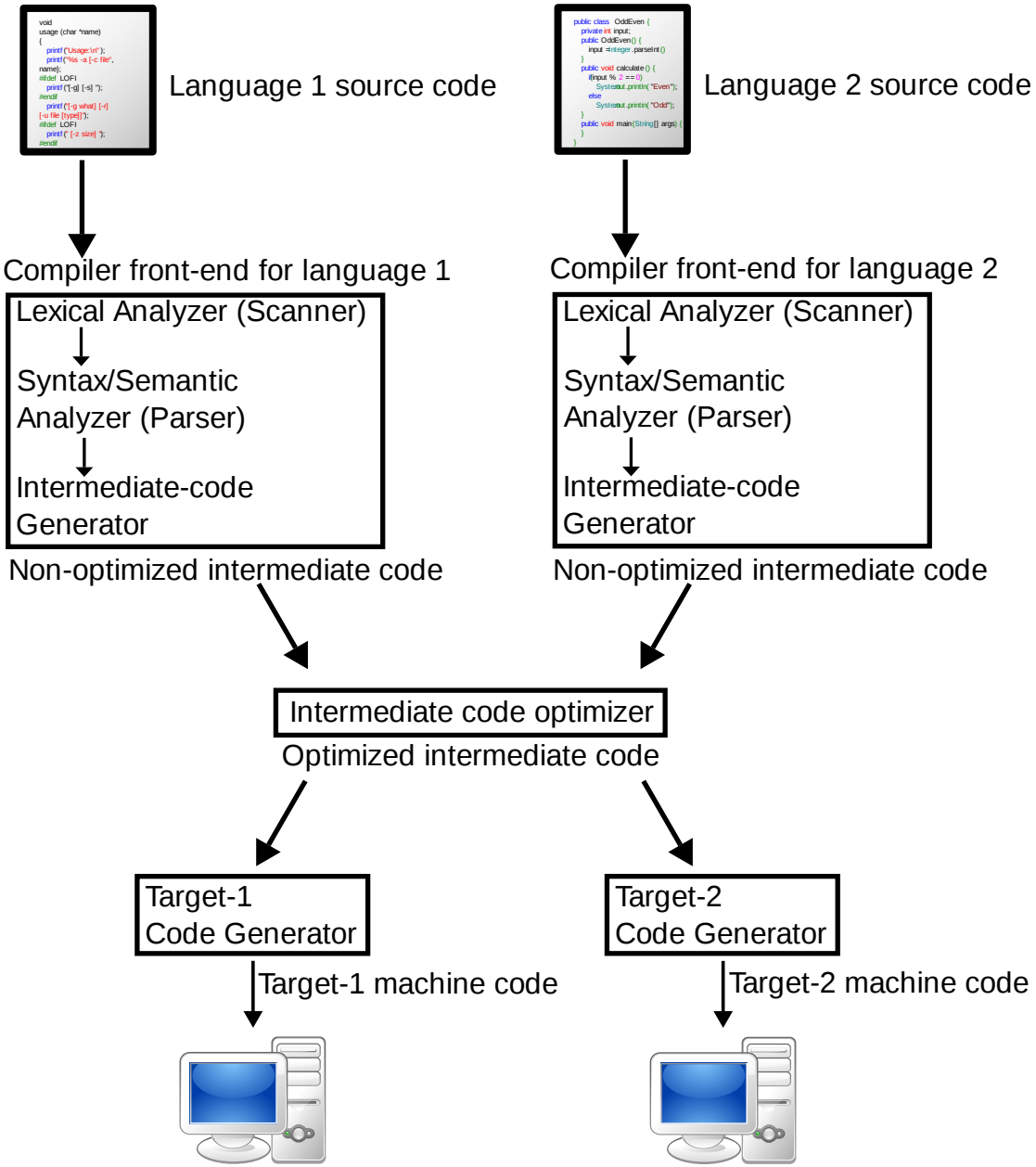
Compilers

- Program that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code)
- Likely to perform many or all of the following operations
  - lexical analysis
  - preprocessing
  - parsing
  - semantic analysis
  - code generation
  - code optimization

Compilers

- Compiler parts:
  - Frontend: checks whether the program is correctly written in terms of syntax and semantics, translation into intermediate representation (IR)
  - Middle-end: optimizations for performance on IR
  - Backend: translation of IR into the target assembly code
- Compiler flags to influence behavior
  - Source code type specifications
  - Target system / CPU
  - Optimization level
  - Inclusion of debug symbols
  - Enable/disable different warnings

Compilers


Language 1 source code
Language 2 source code
Compiler front-end for language 1
Lexical Analyzer (Scanner)
Syntax/Semantic Analyzer (Parser)
Intermediate-code Generator
Non-optimized intermediate code
Compiler front-end for language 2
Lexical Analyzer (Scanner)
Syntax/Semantic Analyzer (Parser)
Intermediate-code Generator
Non-optimized intermediate code
Intermediate code optimizer
Optimized intermediate code
Target-1 Code Generator
Target-2 Code Generator
Target-1 machine code
Target-2 machine code

Compilers

- Some common C, C++ and Fortran compilers
  - Gnu: gcc, g++, gfortran
  - Intel: icc, icp, ifort
  - PGI: pcc, PCC, pgf90
- Commonly used compiler flags
  - `-o <filename>` to specify output filename
  - `-c` To only compile, not link. Result: object file per source file
  - `-Ox` with `x` being a integer to specify optimization level
  - `-g` to include debugging symbols in output
  - `-pg` to include profiling code
  - `-fopenmp` to enable openMP support

Linkers

- Combines object files into larger object file, e.g. executable
- Can be called directly by compiler
- Used as separate step for
  - large projects
  - projects involving different programming languages
  - system wide installed libraries (collections of object files)
- Often not called directly, but through compiler

Compile/Link example commands

Command	Description
`gcc main.c -o main`	Compile and link `main.c` into executable `main`
`gcc -c main.c -o main.o`	Compile `main.c` into object file `main.o`
`ld main.o some.o -o main`	Link `main.o` and `some.o` into executable `main`
`gcc main.c -o main -lm`	Compile and link `main.c` and the math library into executable `main`

Makesystem

Makesystem

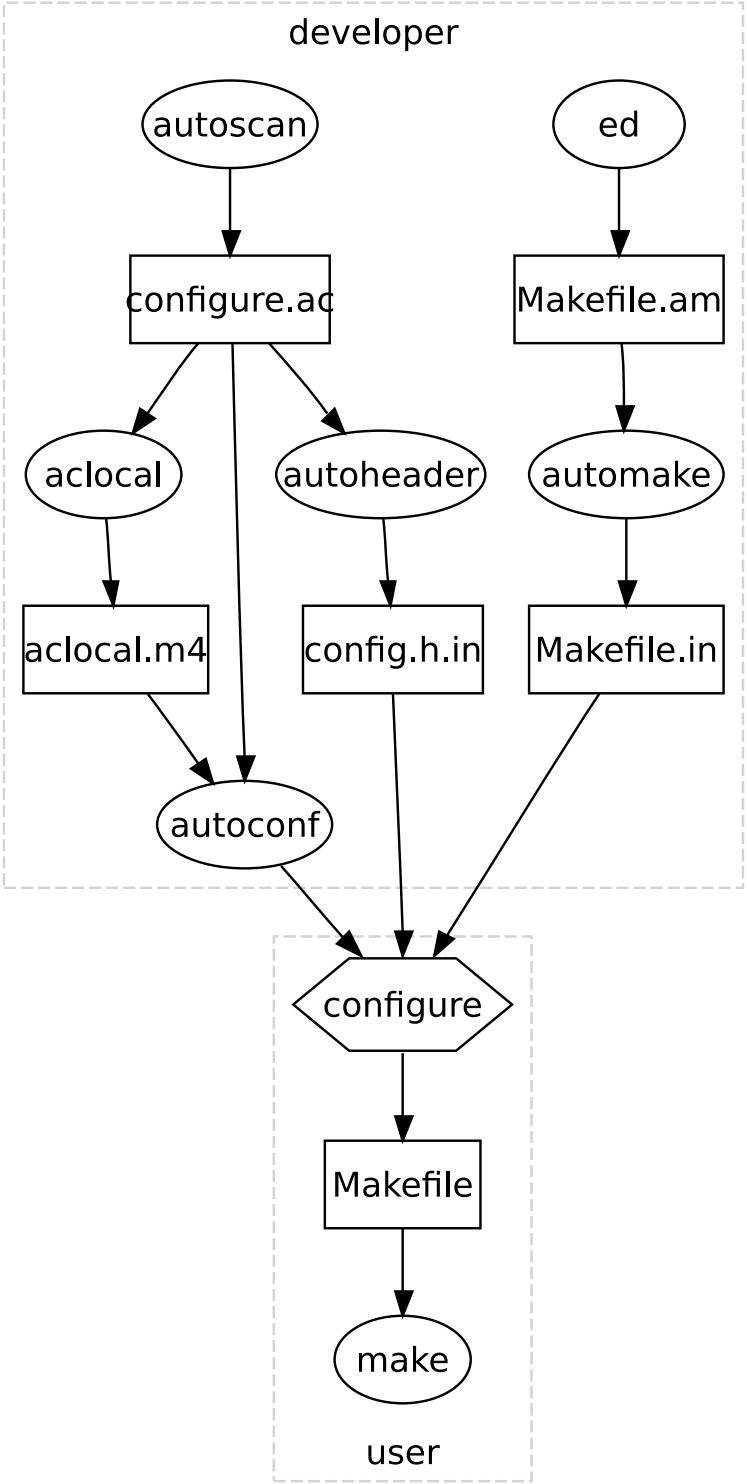
- Utility that automatically builds executable programs and libraries from source code
- Reading files called Makefiles which specify how to derive the target program
- Builds build chain from dependencies between parts of a project
- Easy to introduce bugs in large projects
- Makefiles often built by other tools, for large projects

Buildsystem

- Makefiles do have to be adapted to environment, compilers
- Typical script to generate Makefile from Makefile.in: configure
  - tests various system properties
  - sets defines appropriately in Makefile
- Usually first script of build system a user executes

Buildsystem

Gnu Build System



Visualizing / gnuplot

Visualization

- Simulation results need to be analysed
- Visualization is very intuitive analysis method
- Many and often quite different possibilities
  - different dimentionality, e.g. 1D plots vs. 3D videos
  - different data formats
  - different visualization tools
- Ranges from 1D plots of kByte of data to 3D rendering of TBytes

Visualization, gnuplot

- Simple 1D/2D plotting tool
- Command line interface
- Text input (or anything which can be on-the-fly be converted to text)
- Many output formats (screen, pdf, eps, latex, png, ...)
- Very customizable, but no shallow learning curve
- Mostly used for debugging or final 1D graphs for papers

Visualization, gnuplot

Command	Effect
`plot [0:10] sin(x)`	Plots $\sin(x)$ in the x-range $[0, 10]$
`plot 'data.dat' using 1:3`	Plots columns 1 and 3 of file `data.dat` as x and y
`plot 'data.dat' with lines`	Plots columns 1&2 using lines
`plot '<bzcat data.dat.bz2'`	Decompresses file `data.dat.bz2` on the fly and plot content
`set terminal postscript eps`	Specify eps output format
`set output "plot.eps"`	Set output filename to `plot.eps`

Visualization, gnuplot

Consider the following example data file ("data") where the first, second, third, and fourth columns hold the x, y, min x, and max x values, respectively:

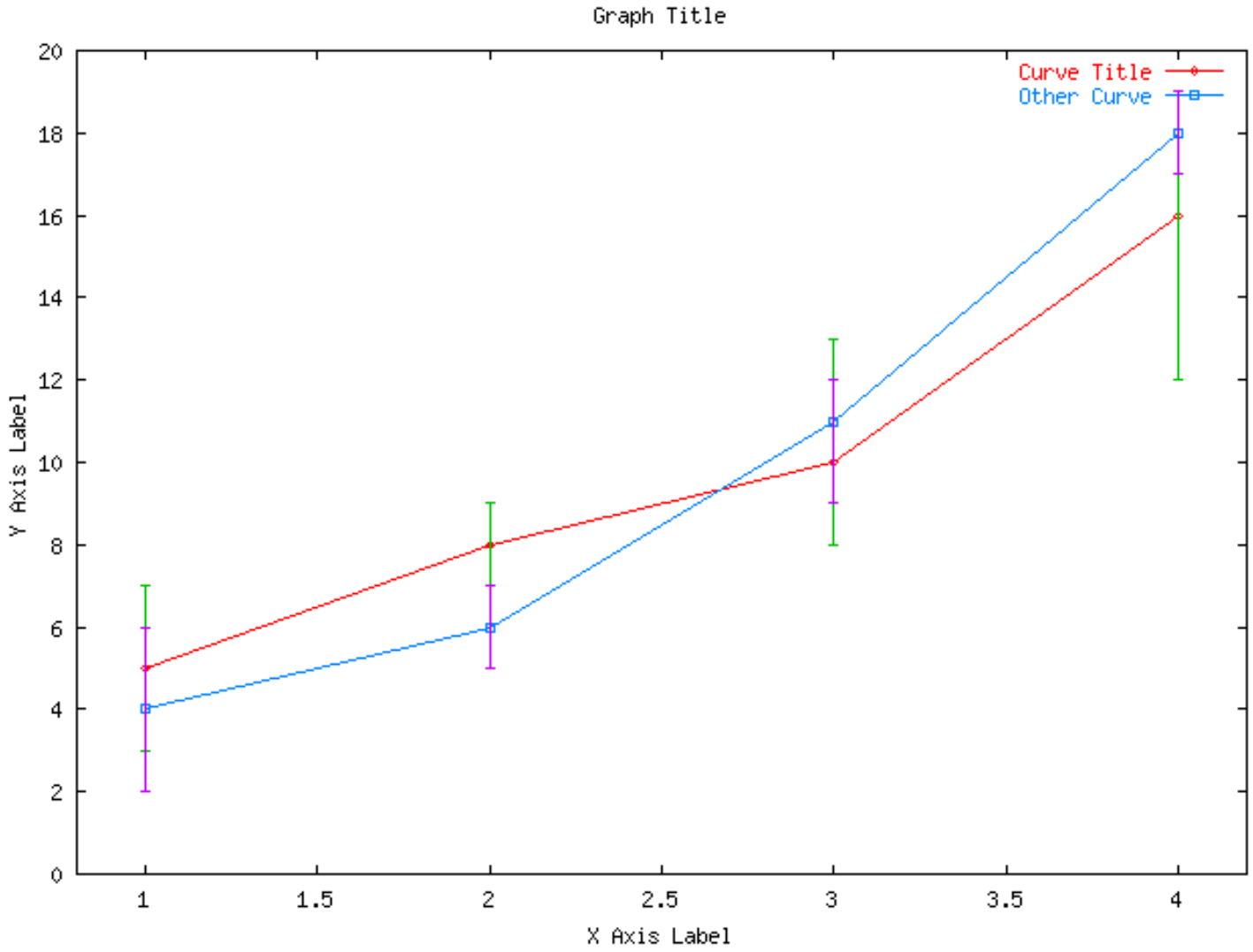
```
1 5 3 7 4
2 8 5 9 6
3 10 8 13 11
4 16 12 19 18
```

You can plot this with following gnuplot command script:

```
set title "Graph Title"
set xlabel "X Axis Label"
set ylabel "Y Axis Label"
set term gif
set output "2D.gif"
# To make postscript do
# set term postscript eps
# set output "2D.eps"
set data style lp
plot [.8:4.2] "2D.data" using 1:2 t "Curve Title", \
              "2D.data" using 1:2:3:4 notitle with errorbars lt 1 ps 0, \
              "2D.data" using 1:5 t "Other Curve", \
              "2D.data" using 1:5:6:7 notitle with errorbars lt 1 ps 0
```

This generates a gif file "2D.gif" ...

Visualization, gnuplot



Coursework

Coursework

- Write a simple program and Makefile that links to an external library (the math library) and use one of those library functions.
- Compile and run on head node of one TeraGrid resource.
- Optional: output a sin table (value pairs of $(x, \sin(x))$ to a file and produce an eps plot with gnuplot.
- Document which steps you had to follow (short)
- Provide source and make file, send files to (sci-comp-instructors@cct.lsu.edu)
- Log in to every of the following systems
  - login-abe.ncsa.teragrid.org
  - tg-login.lonestar.tacc.teragrid.org
  - tg-login.ranger.tacc.teragrid.org
  - queenbee.loni-lsu.teragrid.org
  - tg-steele.purdue.teragrid.org
  - mss.ncsa.teragrid.org

Due: Tue Aug 31st