

Overview and Tools

Advanced Programming in the UNIX Environment

Chun-Ying Huang <chuang@cs.nctu.edu.tw>

Outline

Prepare your UNIX environment

A Brief Introduction to the UNIX environment

Fundamental UNIX programming practices

Tools

Prepare Your UNIX Environment

Use Ubuntu Linux (Debian-based) as examples

Recommended minimum set of Ubuntu packages

- gcc, g++, gdb, make
- manpages-dev, manpages-posix, manpages-posix-dev
- ```
$ sudo apt-get install gcc gcc-multilib g++ gdb make
 manpages-dev manpages-posix manpages-posix-dev
 yasm nasm libcapstone-dev
```
- <https://help.ubuntu.com/11.04/serverguide/apt.html.en> (Ubuntu Linux)
- <https://wiki.archlinux.org/index.php/pacman> (Arch Linux)

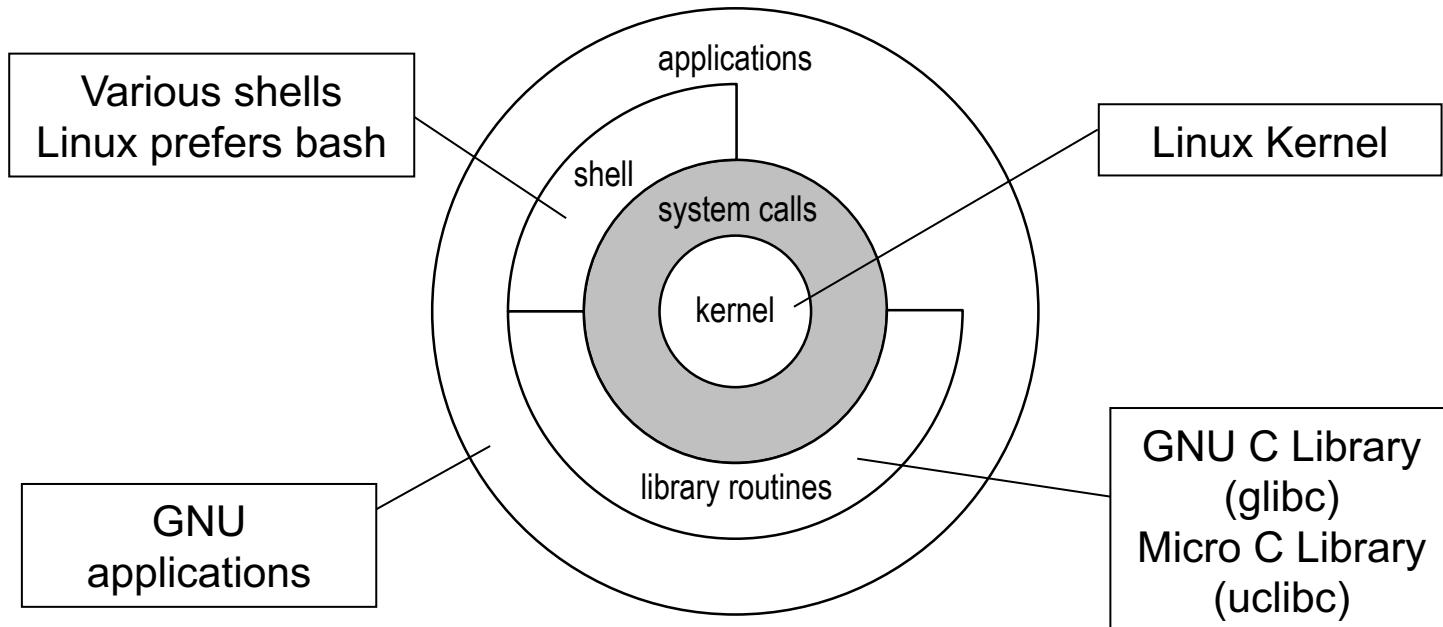
Virtual machine settings

- Virtualbox
- Dual network interfaces
  - Primary: NAT (for Internet access)
  - Secondary: Host-Only (for host-to-host communication, e.g., ssh connection from local to VM)

# A Brief Introduction to the UNIX Environment

---

# UNIX Architecture

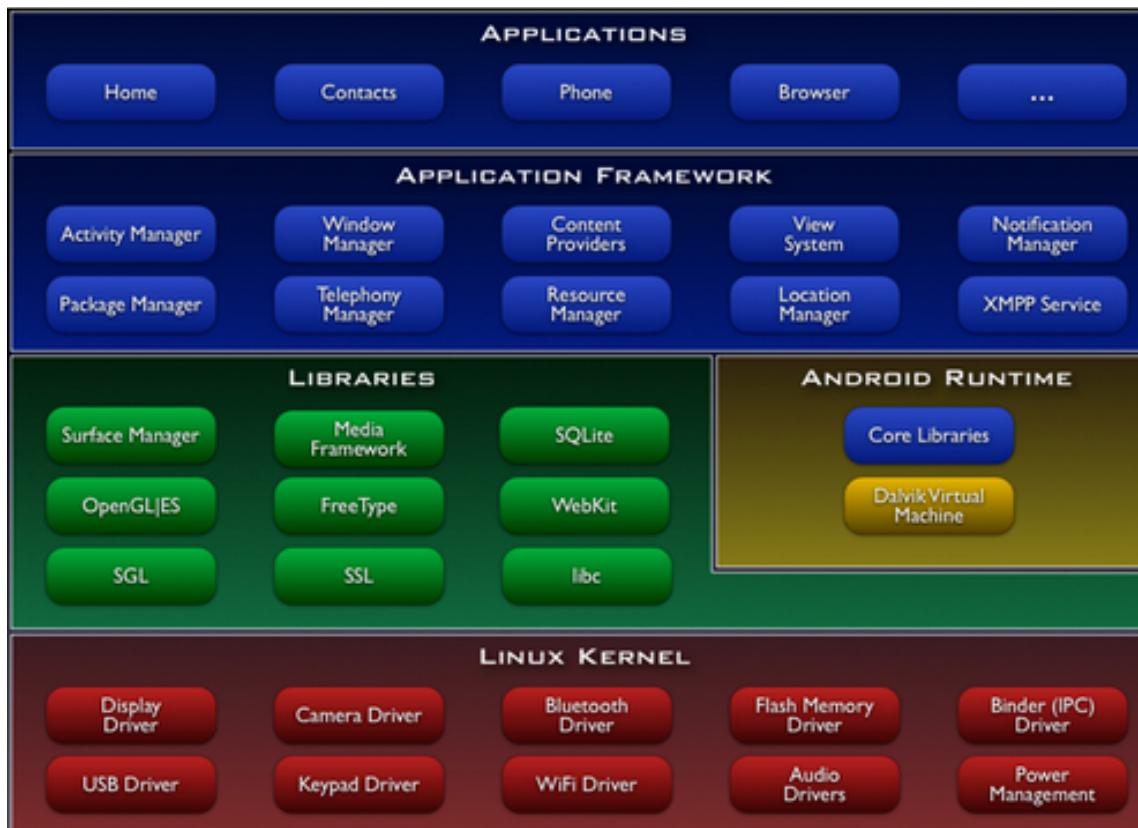


## List of Linux system calls

- <http://man7.org/linux/man-pages/man2/syscalls.2.html>
- (x86\_64) [https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall\\_64.tbl](https://github.com/torvalds/linux/blob/master/arch/x86/entry/syscalls/syscall_64.tbl)
- (x86) <https://syscalls.kernelgrok.com/>
- (x86\_64) [http://www.cs.utexas.edu/~bismith/test/syscalls/syscalls64\\_orig.html](http://www.cs.utexas.edu/~bismith/test/syscalls/syscalls64_orig.html)
- (multiple) <https://w3challs.com/syscalls/>

# UNIX Architecture (Cont'd)

An Android Example



# Booting Process

---

OS loader (e.g., grub)

- Loads kernel and an (optional) initial RAM disk into the memory

Kernel initializes system hardware components

Launch the first process

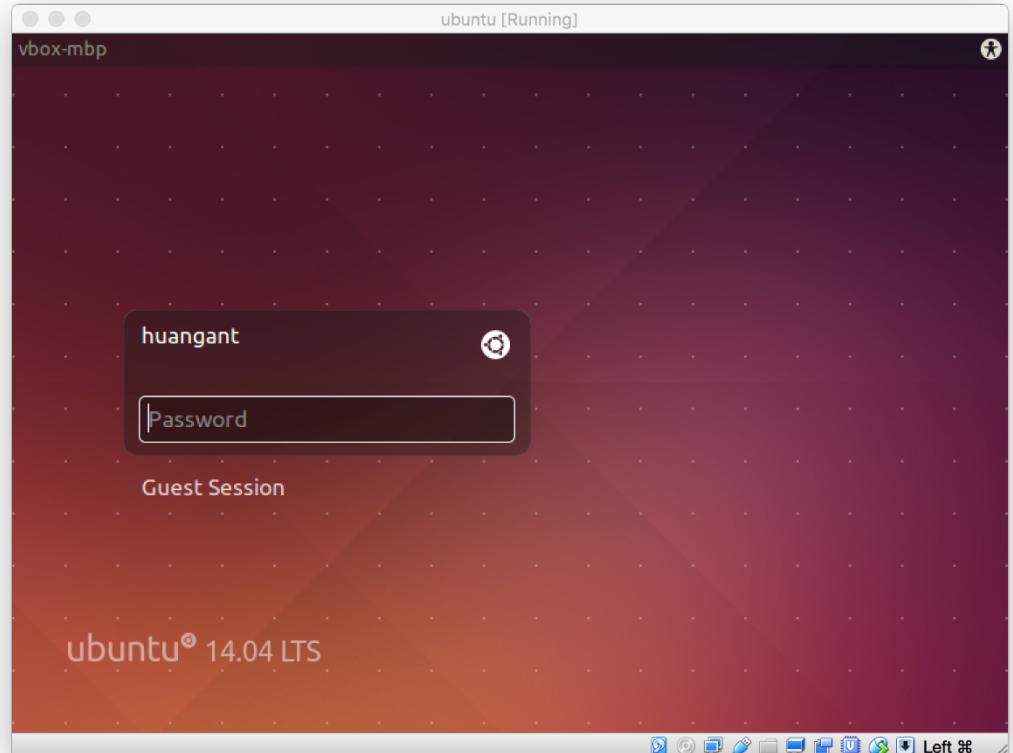
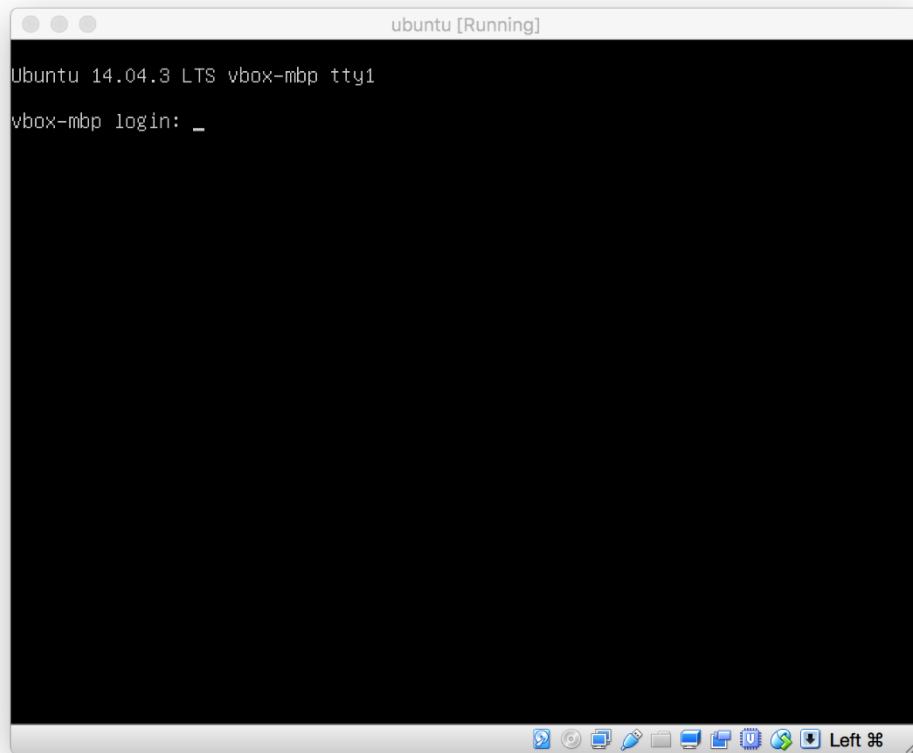
- /sbin/init, /etc/init, /bin/init, and finally /bin/sh
- Modern Linux systems have replaced init with systemd

The init process brings up the rest of everything

- Mount file systems
- Setup networks
- Launch services
- Provide the login interface

# Your First Impression

---



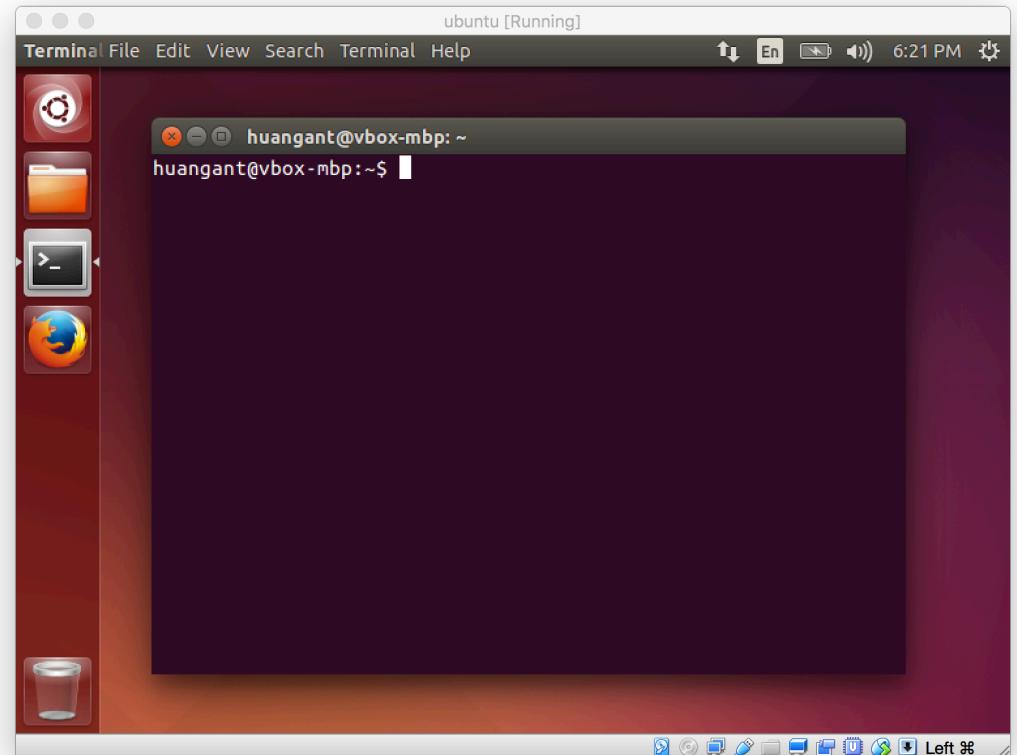
# Your Second Impression

---

ubuntu [Running]

```
Ubuntu 14.04.3 LTS vbox-mbp tty1
vbox-mbp login: huangant
Password:
Last login: Tue Nov 18 15:45:06 CST 2014 on tty1
Welcome to Ubuntu 14.04.3 LTS (GNU/Linux 3.13.0-39-generic i686)

 * Documentation: https://help.ubuntu.com/
huangant@vbox-mbp:~$ _
```



# File System Architecture

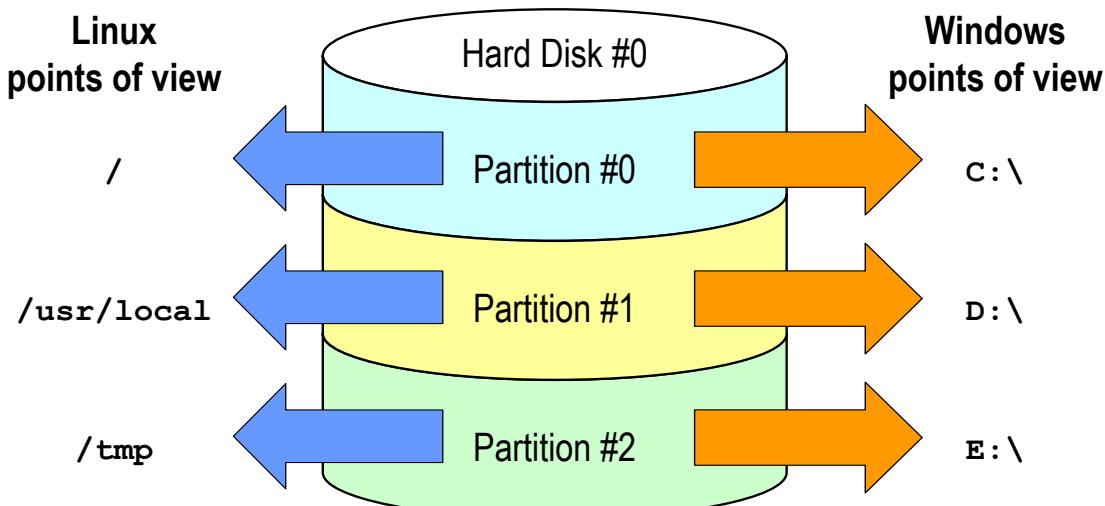
Hierarchical arrangement of directories and files

Everything starts from the "root directory" (/)

The "mount" program

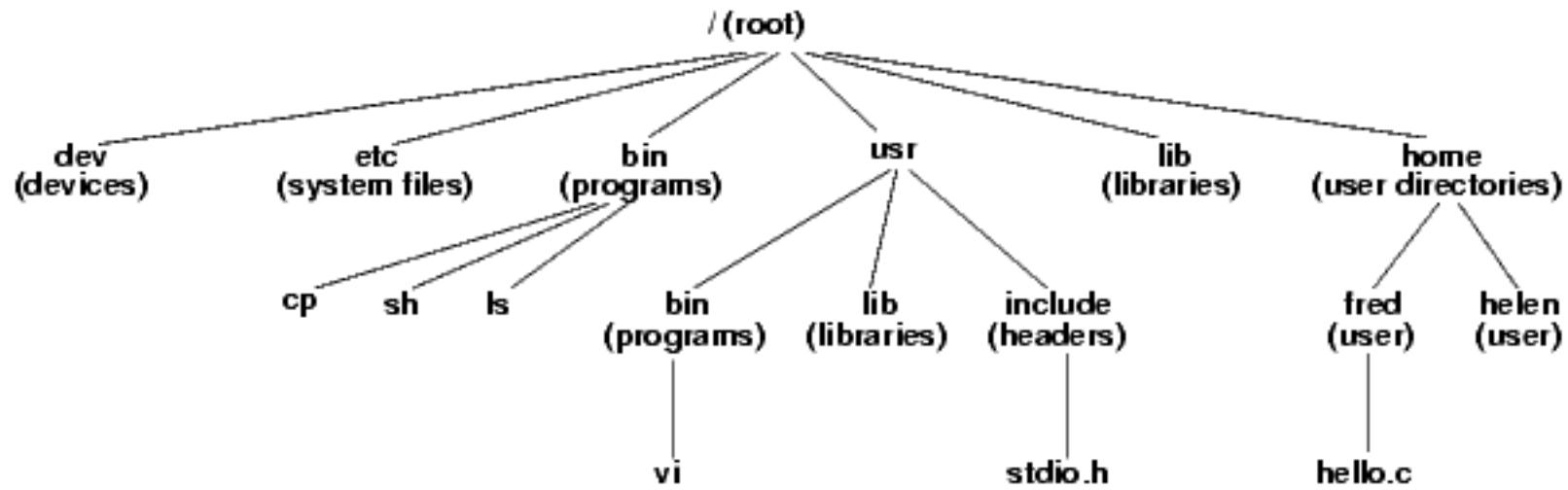
Filenames (and commands) are usually **case-sensitive**

- Mac OS X's default file system (HFS) is an exception



# File System Architecture (Cont'd)

---



# Working with UNIX Commands

---

## Simple commands

- ls: list files
- mkdir: make directory
- cd: change directory
- pwd: print working directory
- rmdir: remove directory
- cp: copy
- mv: move
- rm: remove
- cat: concatenate and print
- less (or more): page splitter
- echo: print a string
- date: print or set the date and time
- env: print out all environment vars
- touch: change file time
- tar: archive tool

# Working with UNIX Commands (Cont'd)

---

There are \*\*a lot of\*\* UNIX commands, and it is impossible to cover all of them in this course!!!

Built-in commands: Provided by the login shell

Other commands: Binaries installed by the system administrators

- Often placed in standard locations
- For examples: /bin, /sbin, /usr/bin, /usr/sbin, /usr/local/bin, /usr/local/sbin
- Binaries are searched according the directories listed in the PATH environment

The Linux standard base (LSB)

- What tools and libraries are mandatory for a Linux operating system
- Official: <http://www.linuxfoundation.org/collaborate/workgroups/lsb>

The (Linux) filesystem hierarchy standard (FHS), is one part of the LSB

- Recommended locations to placed your files
- Official: <http://www.linuxfoundation.org/collaborate/workgroups/lsb/fhs>

# Common Notations in UNIX Documents

---

Run command as a regular user

- % *command* ... (sometimes we use \$ instead of %)

Run command as a privileged user (super user, or root)

- # *command* ... (don't be confused with comments)

Special symbols for command arguments

- Square brackets [ ]: optional part, e.g., cat [filename]
- Dots ...: Multiple arguments are allowed, e.g., cat [file ...]
- Dash - or --: Usually options for a command, e.g.,
  - Part of options for the ls command: -a, --all, --color, -F, ...
  - Single dash options may be aggregated: -aF

# Redirection and Pipe

---

## Redirection

- Outputs of a command can be stored in a file
  - % echo Hello, World! > a
  - % echo Hello, World! >> a
- File content can be used as inputs to a command
  - % cat < a

## Pipe

- Outputs of a command can be inputs of another command
  - % echo Hello, World! | cat
  - % cat hello.c | less
- Pipe can be chained
  - % echo Hello, World! | tr a-z A-Z | cat

# Read the Manual Pages

---

## The man(1) command

- The command you must know in the UNIX world!

Manual pages for commands, system calls, library functions, kernel routines, ...

## Basic usage

- \$ man [section] page
- \$ man -k regexp

## A common notation – page (section)

- Examples:

ls(1), man(1), read(2), crypt(3), tty(4), shadow(5), printf(1), printf(3), ...

# Read the Manual Pages (Cont'd)

---

## Manual page sections

1. User commands
2. System calls
3. C library functions
4. Devices and special files
5. File formats and conventions
6. Games et al.
7. Miscellaneous
8. System administration tools and daemons

# Course Sample Codes

---

Available from the course web site

## UNIX Programming

- Course web site -> Downloads -> Sample code download
- Instructor’s version: modified version with a Makefile
- Text-book’s version: unmodified from the text-book authors

Password protected

Sample commands to download, extract, and build the sample codes

```
$ wget --user unix1XX --password PASSWORD
 http://people.cs.nctu.edu.tw/~chuang/courses/unixprog/resources/inclass-20180508.tar.gz
$ wget --user unix1XX --password PASSWORD
 http://people.cs.nctu.edu.tw/~chuang/courses/unixprog/resources/textbook-20180326.tar.gz
$ tar xzf inclass-20180508.tar.gz
$ tar xzf textbook-20180326.tar.gz
$ cd unix_prog
$ make
```

# Fundamental UNIX Programming Practices

---

# The "Hello, World." Sample

---

```
#include <stdio.h>

int main() {
 printf("Hello, World.\n");
 return 0;
}
```

Compile and run "Hello, World." in the UNIX environment

- \$ gcc hello.c // this generates **a.out**
- \$ ./**a.out**
- \$ gcc hello.c -o hello // this generates **hello**
- \$ ./**hello**

# Return from the main() Function

---

## Return value of the main() function

- It is actually a one byte value
- Return zero: the value indicates 'True', or no problem
- Return non-zero values: the values indicate 'False', or error
- Can be used to determine program execution status ... the 'bash -e' option!

## Read return values from your program

- Run '`echo $?`' immediately right after your program execution

## Some examples (see return.c and testret.sh)

- `return 0;`
- `return 10;`
- `return -10;`

# Return from the main() Function (Cont'd)

---

Shell's short cut branch

- Break an evaluation when the final result is known

Boolean OR (||) – Stop evaluation when a condition is true

- \$ ./return 0 || echo 'A'
- \$ ./return 1 || echo 'B'
- \$ true || echo 'A'
- \$ false || echo 'B'

Boolean AND (&&) – Stop evaluation when a condition is false

- \$ ./return 0 && echo 'C'
- \$ ./return 1 && echo 'D'
- \$ true && echo 'C'
- \$ false && echo 'D'

# Handle Program Options

---

```
int main(int argc, char *argv[]) {
 int i;
 for(i = 0; i < argc; i++)
 printf("'"s' ", argv[i]);
 printf("\n");
 return 0;
}
```

What will be the outputs?

- \$ ./args
- \$ ./args a b c d
- \$ ./args "a b c d"
- \$ ./args 'a b c d'
- \$ ./args "home = \$HOME"
- \$ ./args 'home = \$HOME'

# Handle Program Options (Cont'd)

---

The getopt(3) and getopt\_long(3) style options

Built-in standard option handling routines in many UNIX platforms

getopt(3) reads dash plus single character options (short options)

- Options can be aggregated
- For example, -a -b is equivalent to -ab

getopt\_long(3) also reads double-dash plus key word options (long options)

- For example, --all, --color

```
$ ping
```

```
Usage: ping [-aAbBdDfhLn0qrRUvV] [-c count] [-i interval] [-I interface]
 [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
 [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
 [-w deadline] [-W timeout] [hop1 ...] destination
```

# getopt(3)

---

```
int getopt(int argc, char * const argv[], const char *optstring);
```

- argc: the argc parameter received by the main function
- argv: the argv parameter received by the main function
- optstring: list of valid option characters (usually colons and alphabets)
- Add a colon (:) right after an option character indicates that the option requires an additional argument
- Common return value of getopt(3)
  - -1: No more options
  - Colon (:) or question mark (?): Invalid option encountered
- Global variables
  - optind: An integer stores the number of arguments consumed by getopt(3)
  - optarg: A string points to the additional argument of the current option (if : is given)
- Examples: see getopt.c

# UNIX Time Representations

---

Wall clock time: `time_t`, in second unit

- Number of seconds elapsed from 00:00:00, January 1st, 1970 UTC (the "epoch")
- It is often a 32-bit signed integer
- Will be overflowed after 03:14:07 January 19th, 2038 – The year 2038 problem!

High precision time: `struct timeval`, in microsecond unit

- Basically `time_t` plus a microsecond precision timestamp

CPU time: `clock_t`, in CPU-ticks unit

- The `CLOCKS_PER_SEC` constant
- POSIX requires `CLOCKS_PER_SEC` to be 1,000,000 independent of the actual clock resolution

From `select(2)`:

```
struct timeval {
 long tv_sec; // seconds
 long tv_usec; // microseconds
};
```

# UNIX Time Representations (Cont'd)

---

time(3) function: Get time in time\_t format

```
time_t time(time_t *t);
```

gettimeofday(2) function: Get time in struct timeval format

- The use of tz is obsoleted, it should be NULL

```
int gettimeofday(struct timeval *tv, struct timezone *tz);
```

clock(3) function: Get time in clock\_t format

```
clock_t clock(void);
```

# Measure Program Performance

---

A simple metric: Program running time

A simple example: the time command

```
$ time sleep 10
real 0m10.003s
user 0m0.000s
sys 0m0.003s
```

Real time, user time, and sys time

How to get these numbers?

- gettimeofday(3): get wall clock time in microsecond precision (in timeval format)
- clock(3): get CPU ticks (user + sys)
- getrusage(3): get CPU time (in timeval format)

# Measure Program Performance (Cont'd)

---

Basic idea

```
t0 = get_the_current_timestamp();

// The codes we want to measure ...
Do something ...
```

```
t1 = get_the_current_timestamp();
```

```
Compute and output (t1 - t0)
```

See examples: testtime.c and rusage.c

# Error Handling

---

## Check function return values

- (Integer) Zero or positive values: return without errors
- (Integer) Negative values (usually -1): return with errors
- (Pointer) Non-NULL: return without errors
- (Pointer) NULL: return with errors
- This is applicable for most of the C library functions

## What kinds of error?

- Determine using the `errno` variable.
- A global variable built in C library
  - It is not thread-safe! (*in the past*)
  - But not a problem if your system supports thread local storage (TLS)
- Check it right after receiving an error return value

## List of error codes

- See `errno (3)` manual pages

# Display Errors

---

## Required headers and declarations

```
#include <stdio.h> // for perror

#include <string.h> // for strerror

#include <errno.h> // errno variable, and definitions for error codes
```

## Convert an error number to a human-readable string

- strerror
- perror

```
printf("error = %s\n", strerror(errno));
perror("some prefix");
```

# Error Recovery

---

## Fatal errors

- No way to recovery
- Show error messages, log, and then exit

## Non-fatal errors

- May be temporary errors
- Delay for a short time and then retry
- Examples
  - EAGAIN, ENFILE, ENOBUFS, EWOULDBLOCK, ...

# Tools

---

# The Compiler

---

gcc – GNU C Compiler

g++ – GNU C++ Compiler

## Frequently used options

- -S: do not compile, generate assembly only (output to .s)
- -E: do not compile, perform preprocessing only (output to stdout)
- -c: compile only, do not link
- -g: embed debugging information
- -Wall: turn on all warnings
- -I: add include path, e.g., -I/usr/local/include
- -l: link with a library, e.g., -lxxx will link with a library named libxxx.a
- -L: add library path, e.g., -L/usr/local/lib

# Compile a Single Source Code

Compile and generate the executable binary

- % g++ hello.cpp (the output will be **a.out**)
  - % g++ hello.cpp -o hello (the output will be **hello**)

## Execute the executable

- % ./a.out (or ./hello)
  - Do not miss the **./** prefix

# Try -E and -S options

# Compile Multiple Source Code Files

---

Suppose you have s1.cpp, s2.cpp, and s3.cpp

Strategy #1

- `g++ s1.cpp s2.cpp s3.cpp -o output` (generates **output**)

Strategy #2

- `g++ -c s1.cpp` (generates **s1.o**)
- `g++ -c s2.cpp` (generates **s2.o**)
- `g++ -c s3.cpp` (generates **s3.o**)
- `g++ s1.o s2.o s3.o -o output` (generates **output**)

Which one would be better?



# Linking C and C++ Files (1/3)

---

Suppose we have two source code files, **a.c** and **b.cpp**

a.c:

```
int b();
int main() {
 return b();
}
```

b.cpp:

```
int b() {
 return -1;
}
```

We then compile and link the two files:

- gcc -c a.c (generates a.o)
- g++ -c b.cpp (generates b.o)
- g++ -o test a.o b.o (does it work?) 

# Linking C and C++ Files (2/3)

---

Let's check what we have in the object codes

We can use the **nm** tool to dump symbols

```
$ nm a.o
```



U b

0000000000000000 T main

```
$ nm b.o
```

0000000000000000 T \_Z1bv

Why?

# Linking C and C++ Files (3/3)

---

We have to modify b.cpp if a function will be called from a C program

b.cpp:

```
#ifdef __cplusplus (only needs for a C++ compiler)
extern "C" { (declare that everything within the scope)
int b(); (should be handled as C symbols, not C++)
}
#endif

int b(int n) {
 return n;
}

int b() { $ nm b.o
 return b(-1); 000000000000000c T b
} 0000000000000000 T _Z1bi
```

# Make and Makefile

---

# Outline

---

Why make and Makefile?

The make command

The Makefile

Examples

# Why make and Makefile

---

## Project management

- Simplify build process
- Manage project dependencies

## A common scenario

- Build a program with multiple source files

## Steps

- Write rules in a file named *Makefile*
- Run the *make* command
  - By default, make run *the first* rule in the makefile

# The make Command

---

Simply type ‘make’ in the command prompt

- \$ make
- Or alternatively, specify a target rule: \$ make clean

Common options

- -C {dir}: switch to the given directory and run the make command
- -f {makefile}: use a non-default makefile
- -I {dir}: specify include directory search path
- -j {n}: allow simultaneously jobs (commands)
- For the details, see the man page!

# The Makefile

---

Rule definitions

Variable definitions

Automatic variables

Special rules

Pattern rules

# Rule Definitions

---

## General format

- rulename: dependencies (or prerequisites)  
 (tab) rules

Rulename – the target to be built

## Dependencies

- Prerequisites required to build the target
- Separated by spaces

## Rules

- Commands to build the target

# Rule Definitions (Cont'd)

---

Comments: start with a pond sign (#)

Split a single line into multiple lines: back slash (\)

- See (external) examples

# An Example

---

```
test: test1.c test2.c # comment
 gcc -c test1.c
 gcc -c test2.c
 gcc -o test test1.o test2.o
```

This is *not* a good example

Both the test1.c and the test2.c files are compiled if either one of them is modified – refine it later ...

# Variable definitions

---

## Common usage

- Set: VARNAME=value
- Use: \$(VARNAME)

## Create variables

- CC = gcc
- CXX = g++
- CFLAGS = -I. -Wall



## Use the variables

- \$(CC) -c test.c \$(CFLAGS)
- Note that a nonexistence variable produces an empty string

# Automatic Variables

---

\$@: The target file name 

\$<: The name of the first prerequisite

\$?: The name of all prerequisites that are *newer* than the target

\$^: The name of all prerequisites. *Duplicated entries will be removed*

\$+: Like \$^, but duplicated entries will *not* be removed

# A Refined Example

---

```
CC = gcc
```

```
CFLAGS = -g -Wall
```

```
✉ .c.o: # old-fashioned!
 $(CC) -c $< $(CFLAGS)
```

```
test: test1.o test2.o
 $(CC) -o test $^
```

This one is better!

Only modified objects will be re-built

# Special Rules

---

## .SUFFIXES (old fashioned!)

- Add non-default suffixes (filename extensions)
- Example

```
.SUFFIXES: # (remove all)
```

```
.SUFFIXES: .asm .inc
```

## .PHONY

- Don't check if a target is an existing file
- Example

```
.PHONY: all clean
```





# Pattern Rules – The % Symbol

---

Definition of “stem”: the text between the prefix and the suffix

Remember the old-fashioned “.c.o:” rule?

It is equivalent to

`%.o: %.c` 

The new style provides much more flexibilities

# External Examples

---

## Sample codes

- sample codes/Makefile
- sample codes/Make.rules
- sample codes/lib/Makefile

## A complete reference to make and Makefile

- The make manual page
- <http://www.gnu.org/software/make/manual/>

# Debug with GDB

---

# GDB – Quick Introduction

---

A command line based (interactive) debugger

All source codes must be compiled with the -g option!

Example #1

- \$ gcc **-g** test.c

Example #2

- Makefile
- See CFLAGS

```
GCC = gcc
CFLAGS = -g -Wall

.c.o: # old-fashioned!
 $(GCC) -c $< $(CFLAGS)

test: test1.o test2.o
 $(GCC) -o test $^
```

# The First Impression

---

```
$ gdb a.out # a.out is the program executable
```

```
GNU gdb (GDB) 7.10.1
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...done.
(gdb) _
```

# Compiled without -g Option

---

```
$ gdb a.out # a.out is the program executable
```

```
GNU gdb (GDB) 7.10.1
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from a.out...(no debugging symbols found)...done.
(gdb) _
```

# Basic Commands

---

Show source codes

- list [ line # | function | file:line # | file:function ]

Start to debug a program

- run [arguments ...]

Run the next command

- next (will *not* enter a function) 
- step (will enter a function)

Display

- print

# Breakpoints

---

## Set breakpoints

- `break [ line # | function | file:line # | file:function ]`

## Delete breakpoints

- `clear [ line # | function | file:line # | file:function ]`

## Show breakpoints

- `info breakpoints`

## Run until a breakpoint is reached

- `continue`

# Sample Source Code

---

Source code: hello.c

```
1: #include <stdio.h>
2:
3: int
4: main() {
5: int i;
6: char hello[] = "Hello, World!\n";
7: char *ph = hello;
8: for(i = 0; ph[i]!='\0'; i++) {
9: putchar(ph[i]);
10: }
11: return 0;
12: }
```

# Compile, Load, and Run

---

```
$ gcc -g bug.c -o bug
$ gdb bug
(gdb) run
Starting program: /raid/home/chuang/tmp/bug
Hello, World!
```

```
Program exited normally.
(gdb)
```

# List Source Codes

---

```
(gdb) list 1
1 #include <stdio.h>
2
3 int
4 main() {
5 int i;
6 char hello[] = "Hello, World!\n";
7 char *ph = hello;
8 for(i = 0; ph[i]!='\0'; i++) {
9 putchar(ph[i]);
10 }
(gdb)
```

# Set Breakpoints and Run

---

```
(gdb) b 8
Breakpoint 1 at 0x8048485: file bug.c, line 8.
(gdb) run
Starting program: /raid/home/chuang/tmp/bug

Breakpoint 1, main () at hello.c:8
8 for(i = 0; ph[i]!='\0'; i++) {
(gdb) n
9 putchar(ph[i]);
(gdb) print i
$1 = 0 # print a variable
(gdb) print ph[i]
$2 = 72 'H' # print another variable
(gdb) print putchar(ph[i])
$3 = 72 # run a function, may also consider 'call'
(gdb) print printf("%c\n", ph[i])
HH # note that we are using buffered I/O
$4 = 2
(gdb)
```

# Debug a Crashed Program

---

Given another  
buggy program ...

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 void walk(int depth) {
6 int *p = 0;
7 printf("%d\n", depth);
8 if(rand()%5==0) *p = depth;
9 else walk(depth+1);
10 return;
11 }
12
13 int main() {
14 srand(time(0));
15 walk(0);
16 return 0;
17 }
```

# Debug a Crashed Program

---

Run the program

```
$ gcc -g bug2.c -o bug2
$./bug2
```

```
0
1
2
3
4
```

*Segmentation fault (core dumped)*

```
$ ls -la bug core
-rwxr-xr-x 1 chuang chuang 8577 2015-02-28 15:58 bug2
-rw----- 1 chuang chuang 155648 2015-02-28 15:58 core
```

# Debug a Crashed Program

---

Load the core and show call stack

```
$ gdb bug2 core
...
Reading symbols from /raid/home/chuang/tmp/bug2...done.
[New Thread 31638]

warning: Can't read pathname for load map: Input/output error.
Reading symbols from /lib/tls/i686/cmov/libc.so.6...(no debugging symbols found)...done.
Loaded symbols for /lib/tls/i686/cmov/libc.so.6
Reading symbols from /lib/ld-linux.so.2...(no debugging symbols found)...done.
Loaded symbols for /lib/ld-linux.so.2
Core was generated by `./bug2'.
Program terminated with signal 11, Segmentation fault.
#0 0x080484c3 in walk (depth=4) at bug.c:8
8 if(rand()%5==0) *p = depth;
(gdb) bt
#0 0x080484c3 in walk (depth=4) at bug2.c:8
#1 0x080484d5 in walk (depth=3) at bug2.c:9
#2 0x080484d5 in walk (depth=2) at bug2.c:9
#3 0x080484d5 in walk (depth=1) at bug2.c:9
#4 0x080484d5 in walk (depth=0) at bug2.c:9
#5 0x08048500 in main () at bug2.c:15
(gdb)
```



# No coredump File Generated?

---

You may have to modify your system configuration

You will need root permission

Take Ubuntu Linux as an example: /etc/security/limits.conf

```
/etc/security/limits.conf
#
#Each line describes a limit for a user in the form:
#
#<domain> <type> <item> <value>
#
...
#*
 soft core 0
* soft core 1000000
#root hard core 100000
#* hard rss 10000
...
...
```

# GDB: Debug w/ Library Symbols (1/3)

---

You will need

- Library source codes
- Library files with debugger symbols

Usually you have to add **-g** option and re-compile the library

It is easier to do that on Ubuntu, because

- It has separated debugger symbol packages for some libraries
- For example, C library: the `libc6-dbg` package
- Look at our course Docker file

# GDB: Debug w/ Library Symbols (2/3)

---

Add directories to look for source codes

- The `directory` keyword, for example
- `directory /glibc-2.27/stdio-common`
- `directory /glibc-2.27/libio`

The `putchar()` function in `bug.c` sample program

- See the next slide and demo

# GDB: Debug w/ Library Symbols (3/3)

```
Terminal — 80x22
[(gdb) b 9
Breakpoint 1 at 0x6f1: file bug.c, line 9.
[(gdb) run
Starting program: /home/chuang/unix_prog/env+tools/bug

Breakpoint 1, main () at bug.c:9
9 putchar(ph[i]);
[(gdb) s
putchar (c=72) at putchar.c:25
25 {
[(gdb) list putchar
20
21 #undef putchar
22
23 int
24 putchar (int c)
25 {
26 int result;
27 _IO_acquire_lock (_IO_stdout);
28 result = _IO_putc_unlocked (c, _IO_stdout);
29 _IO_release_lock (_IO_stdout);
(gdb)
```

# GDB: Debug w/o (Debug) Symbols

---

Scenario #1: We have symbols in the binary, but we don't have debugger symbols

- "info functions" to list symbols
- Set breakpoints based on symbol names
- "layout asm" to see codes (^X A to on/off; ^X O to switch focus)
- "set disassembly-flavor intel" to use Intel syntax

Scenario #2: The executable is stripped

- Suppose the executable is a dynamically linked binary
- "set stop-on-solib-events 1" to stop when a so is loaded/unloaded
- When libc.so is loaded ... break on \_\_libc\_start\_main
- Looking for ptr call, e.g., "call rax"

# GDB v5 reference card

---

SOURCE: GDB-7.2 SOURCE CODE

TYPE 'HELP' TO SEE HELPS IN THE GDB INTERFACE

AVAILABLE ON THE COURSE WEBSITE: <HTTP://SNSL.CS.NTOU.EDU.TW/COURSES/LINUXPROG/DAT/REF/GDB72.PDF>

# GDB QUICK REFERENCE GDB Version 5

## Essential Commands

**gdb** *program [core]* debug *program* [using coredump *core*]  
**b** [*file:*]*line* set breakpoint at *function* [in *file*]  
**run** [*arglist*] start your program [with *arglist*]  
**bt** backtrace; display program stack  
**p** *expr* display the value of an expression  
**c** continue running your program  
**n** next line, stepping over function calls  
**s** next line, stepping into function calls

## Starting GDB

**gdb** start GDB, with no debugging files  
**gdb** *program* begin debugging *program*  
**gdb** *program core* debug coredump *core* produced by  
          *program*  
**gdb --help** describe command line options

## Stopping GDB

**quit** exit GDB; also **q** or **EOF** (eg C-d)  
**INTERRUPT** (eg C-c) terminate current command, or  
send to running process

## Getting Help

**help** list classes of commands  
**help** *class* one-line descriptions for commands in  
          *class*  
**help** *command* describe *command*

## Executing your Program

**run** *arglist* start your program with *arglist*  
**run** start your program with current argument  
      list  
**run ... <inf>outf** start your program with input, output  
      redirected  
**kill** kill running program  
  
**tty** *dev* use *dev* as stdin and stdout for next **run**  
**set args** *arglist* specify *arglist* for next **run**  
**set args** specify empty argument list  
**show args** display argument list  
  
**show env** show all environment variables  
**show env** *var* show value of environment variable *var*  
**set env** *var* *string* set environment variable *var*  
**unset env** *var* remove *var* from environment

## Shell Commands

**cd** *dir* change working directory to *dir*  
**pwd** Print working directory  
**make ...** call “make”  
**shell** *cmd* execute arbitrary shell command string

[ ] surround optional arguments    ... show one or more arguments

## Breakpoints and Watchpoints

**break** [*file:*]*line* set breakpoint at *line* number [in *file*]  
**b** [*file:*]*line* eg: **break** *main.c:37*  
  
**break** [*file:*]*func* set breakpoint at *func* [in *file*]  
**break +offset** set break at *offset* lines from current stop  
**break -offset**  
**break \*addr** set breakpoint at address *addr*  
**break** set breakpoint at next instruction  
**break ... if** *expr* break conditionally on nonzero *expr*  
**cond** *n* [*expr*] new conditional expression on breakpoint  
      *n*; make unconditional if no *expr*  
**tbreak ...** temporary break; disable when reached  
  
**rbreak** [*file:*]*regex* break on all functions matching *regex* [in  
          *file*]  
  
**watch** *expr* set a watchpoint for expression *expr*  
**catch** *event* break at *event*, which may be **catch**,  
          **throw**, **exec**, **fork**, **vfork**, **load**, or  
          **unload**.  
  
**info break** show defined breakpoints  
**info watch** show defined watchpoints  
  
**clear** delete breakpoints at next instruction  
**clear** [*file:*]*fun* delete breakpoints at entry to *fun()*  
**clear** [*file:*]*line* delete breakpoints on source line  
**delete** [*n*] delete breakpoints [or breakpoint *n*]  
  
**disable** [*n*] disable breakpoints [or breakpoint *n*]  
**enable** [*n*] enable breakpoints [or breakpoint *n*]  
**enable once** [*n*] enable breakpoints [or breakpoint *n*];  
          disable again when reached  
  
**enable del** [*n*] enable breakpoints [or breakpoint *n*];  
          delete when reached  
  
**ignore** *n* *count* ignore breakpoint *n*, *count* times  
  
**commands** *n*  
          [**silent**]  
          *command-list* execute GDB *command-list* every time  
          breakpoint *n* is reached. [**silent**  
          suppresses default display]  
**end** end of *command-list*

## Program Stack

**backtrace** [*n*] print trace of all frames in stack; or of *n*  
          frames—innermost if *n*>0, outermost if  
          *n*<0  
**bt** [*n*] select frame number *n* or frame at address  
          *n*; if no *n*, display current frame  
**frame** [*n*] select frame *n* frames up  
          select frame *n* frames down  
  
**up** *n* describe selected frame, or frame at *addr*  
**down** *n* arguments of selected frame  
**info frame** [*addr*] local variables of selected frame  
**info args**  
**info locals**  
**info reg** [*rn*]... register values [for regs *rn*] in selected  
          frame; **all-reg** includes floating point  
**info all-reg** [*rn*]

## Execution Control

**continue** [*count*] continue running; if *count* specified, ignore  
          this breakpoint next *count* times  
  
**step** [*count*] execute until another line reached; repeat  
          *count* times if specified  
**s** [*count*] step by machine instructions rather than  
          source lines  
  
**steipi** [*count*] execute next line, including any function  
          calls  
**si** [*count*] next machine instruction rather than  
          source line  
  
**next** [*count*] run until next instruction (or *location*)  
**n** [*count*] run until selected stack frame returns  
**nexti** [*count*] pop selected stack frame without  
          executing [setting return value]  
**ni** [*count*] resume execution with signal *s* (none if 0)  
resume execution at specified *line* number  
          or *address*  
  
**until** [*location*] evaluate *expr* without displaying it; use  
          for altering program variables

## Display

**print** [*f*] [*expr*] show value of *expr* [or last value \$]  
      according to format *f*  
**p** [*f*] [*expr*] hexadecimal  
**x** signed decimal  
**d** unsigned decimal  
**u** octal  
**o** binary  
**t** address, absolute and relative  
**a** character  
**c** floating point  
**f** like **print** but does not display **void**  
  
**call** [*f*] *expr* examine memory at address *expr*; optional  
**x** [*Nuf*] *expr* format spec follows slash  
      *N* count of how many units to display  
      *u* unit size; one of  
          **b** individual bytes  
          **h** halfwords (two bytes)  
          **w** words (four bytes)  
          **g** giant words (eight bytes)  
**f** printing format. Any **print** format, or  
          **s** null-terminated string  
          **i** machine instructions  
**disassem** [*addr*] display memory as machine instructions

## Automatic Display

**display** [*f*] *expr* show value of *expr* each time program  
          stops [according to format *f*]  
**display** display all enabled expressions on list  
**undisplay** *n* remove number(s) *n* from list of  
          automatically displayed expressions  
  
**disable disp** *n* disable display for expression(s) number *n*  
**enable disp** *n* enable display for expression(s) number *n*  
**info display** numbered list of display expressions

## Expressions

|                              |                                                                      |
|------------------------------|----------------------------------------------------------------------|
| <code>expr</code>            | an expression in C, C++, or Modula-2 (including function calls), or: |
| <code>addr@len</code>        | an array of <code>len</code> elements beginning at <code>addr</code> |
| <code>file::nm</code>        | a variable or function <code>nm</code> defined in <code>file</code>  |
| <code>{type}addr</code>      | read memory at <code>addr</code> as specified <code>type</code>      |
| <code>\$</code>              | most recent displayed value                                          |
| <code>\$n</code>             | <code>n</code> th displayed value                                    |
| <code>\$\$</code>            | displayed value previous to <code>\$</code>                          |
| <code>\$\$n</code>           | <code>n</code> th displayed value back from <code>\$</code>          |
| <code>\$_</code>             | last address examined with <code>x</code>                            |
| <code>\$_-</code>            | value at address <code>\$_</code>                                    |
| <code>\$var</code>           | convenience variable; assign any value                               |
| <code>show values [n]</code> | show last 10 values [or surrounding <code>\$n</code> ]               |
| <code>show conv</code>       | display all convenience variables                                    |

## Symbol Table

|                                |                                                                                                                    |
|--------------------------------|--------------------------------------------------------------------------------------------------------------------|
| <code>info address s</code>    | show where symbol <code>s</code> is stored                                                                         |
| <code>info func [regex]</code> | show names, types of defined functions (all, or matching <code>regex</code> )                                      |
| <code>info var [regex]</code>  | show names, types of global variables (all, or matching <code>regex</code> )                                       |
| <code>whatis [expr]</code>     | show data type of <code>expr</code> [or <code>\$</code> ] without evaluating; <code>ptype</code> gives more detail |
| <code>ptype [expr]</code>      | describe type, struct, union, or enum                                                                              |
| <code>ptype type</code>        |                                                                                                                    |

## GDB Scripts

|                                      |                                                                                               |
|--------------------------------------|-----------------------------------------------------------------------------------------------|
| <code>source script</code>           | read, execute GDB commands from file <code>script</code>                                      |
| <code>define cmd command-list</code> | create new GDB command <code>cmd</code> ; execute script defined by <code>command-list</code> |
| <code>end</code>                     | end of <code>command-list</code>                                                              |
| <code>document cmd help-text</code>  | create online documentation for new GDB command <code>cmd</code>                              |
| <code>end</code>                     | end of <code>help-text</code>                                                                 |

## Signals

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <code>handle signal act</code> | specify GDB actions for <code>signal</code> : |
| <code>print</code>             | announce signal                               |
| <code>noprint</code>           | be silent for signal                          |
| <code>stop</code>              | halt execution on signal                      |
| <code>nostop</code>            | do not halt execution                         |
| <code>pass</code>              | allow your program to handle signal           |
| <code>nopass</code>            | do not allow your program to see signal       |
| <code>info signals</code>      | show table of signals, GDB action for each    |

## Debugging Targets

|                                |                                             |
|--------------------------------|---------------------------------------------|
| <code>target type param</code> | connect to target machine, process, or file |
| <code>help target</code>       | display available targets                   |
| <code>attach param</code>      | connect to another process                  |
| <code>detach</code>            | release target from GDB control             |

## Controlling GDB

|                                                                   |                                                                                                          |
|-------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <code>set param value</code>                                      | set one of GDB's internal parameters                                                                     |
| <code>show param</code>                                           | display current setting of parameter                                                                     |
| Parameters understood by <code>set</code> and <code>show</code> : |                                                                                                          |
| <code>complaint limit</code>                                      | number of messages on unusual symbols                                                                    |
| <code>confirm on/off</code>                                       | enable or disable cautionary queries                                                                     |
| <code>editing on/off</code>                                       | control <code>readline</code> command-line editing                                                       |
| <code>height lpp</code>                                           | number of lines before pause in display                                                                  |
| <code>language lang</code>                                        | Language for GDB expressions ( <code>auto</code> , <code>c</code> or <code>modula-2</code> )             |
| <code>listsize n</code>                                           | number of lines shown by <code>list</code>                                                               |
| <code>prompt str</code>                                           | use <code>str</code> as GDB prompt                                                                       |
| <code>radix base</code>                                           | octal, decimal, or hex number representation                                                             |
| <code>verbose on/off</code>                                       | control messages when loading symbols                                                                    |
| <code>width cpl</code>                                            | number of characters before line folded                                                                  |
| <code>write on/off</code>                                         | Allow or forbid patching binary, core files (when reopened with <code>exec</code> or <code>core</code> ) |
| <code>history ...</code>                                          | groups with the following options:                                                                       |
| <code>h ...</code>                                                |                                                                                                          |
| <code>h exp off/on</code>                                         | disable/enable <code>readline</code> history expansion                                                   |
| <code>h file filename</code>                                      | file for recording GDB command history                                                                   |
| <code>h size size</code>                                          | number of commands kept in history list                                                                  |
| <code>h save off/on</code>                                        | control use of external file for command history                                                         |
| <code>print ...</code>                                            | groups with the following options:                                                                       |
| <code>p ...</code>                                                |                                                                                                          |
| <code>p address on/off</code>                                     | print memory addresses in stacks, values                                                                 |
| <code>p array off/on</code>                                       | compact or attractive format for arrays                                                                  |
| <code>p demangl on/off</code>                                     | source (demangled) or internal form for C++ symbols                                                      |
| <code>p asm-dem on/off</code>                                     | demangle C++ symbols in machine-instruction output                                                       |
| <code>p elements limit</code>                                     | number of array elements to display                                                                      |
| <code>p object on/off</code>                                      | print C++ derived types for objects                                                                      |
| <code>p pretty off/on</code>                                      | struct display: compact or indented                                                                      |
| <code>p union on/off</code>                                       | display of union members                                                                                 |
| <code>p vtbl off/on</code>                                        | display of C++ virtual function tables                                                                   |
| <code>show commands</code>                                        | show last 10 commands                                                                                    |
| <code>show commands n</code>                                      | show 10 commands around number <code>n</code>                                                            |
| <code>show commands +</code>                                      | show next 10 commands                                                                                    |
| <b>Working Files</b>                                              |                                                                                                          |
| <code>file [file]</code>                                          | use <code>file</code> for both symbols and executable; with no arg, discard both                         |
| <code>core [file]</code>                                          | read <code>file</code> as coredump; or discard                                                           |
| <code>exec [file]</code>                                          | use <code>file</code> as executable only; or discard                                                     |
| <code>symbol [file]</code>                                        | use symbol table from <code>file</code> ; or discard                                                     |
| <code>load file</code>                                            | dynamically link <code>file</code> and add its symbols                                                   |
| <code>add-sym file addr</code>                                    | read additional symbols from <code>file</code> , dynamically loaded at <code>addr</code>                 |
| <code>info files</code>                                           | display working files and targets in use                                                                 |
| <code>path dirs</code>                                            | add <code>dirs</code> to front of path searched for executable and symbol files                          |
| <code>show path</code>                                            | display executable and symbol file path                                                                  |
| <code>info share</code>                                           | list names of shared libraries currently loaded                                                          |

## Source Files

|                              |                                                                                   |
|------------------------------|-----------------------------------------------------------------------------------|
| <code>dir names</code>       | add directory <code>names</code> to front of source path                          |
| <code>dir</code>             | clear source path                                                                 |
| <code>show dir</code>        | show current source path                                                          |
| <code>list</code>            | show next ten lines of source                                                     |
| <code>list -</code>          | show previous ten lines                                                           |
| <code>list lines</code>      | display source surrounding <code>lines</code> , specified as:                     |
| <code>[file:]num</code>      | line number [in named file]                                                       |
| <code>[file:]function</code> | beginning of function [in named file]                                             |
| <code>+off</code>            | <code>off</code> lines after last printed                                         |
| <code>-off</code>            | <code>off</code> lines previous to last printed                                   |
| <code>*address</code>        | line containing <code>address</code>                                              |
| <code>list f,l</code>        | from line <code>f</code> to line <code>l</code>                                   |
| <code>info line num</code>   | show starting, ending addresses of compiled code for source line <code>num</code> |
| <code>info source</code>     | show name of current source file                                                  |
| <code>info sources</code>    | list all source files in use                                                      |
| <code>forw regex</code>      | search following source lines for <code>regex</code>                              |
| <code>rev regex</code>       | search preceding source lines for <code>regex</code>                              |

## GDB under GNU Emacs

|                        |                                                    |
|------------------------|----------------------------------------------------|
| <code>M-x gdb</code>   | run GDB under Emacs                                |
| <code>C-h m</code>     | describe GDB mode                                  |
| <code>M-s</code>       | step one line ( <code>step</code> )                |
| <code>M-n</code>       | next line ( <code>next</code> )                    |
| <code>M-i</code>       | step one instruction ( <code>stepti</code> )       |
| <code>C-c C-f</code>   | finish current stack frame ( <code>finish</code> ) |
| <code>M-c</code>       | continue ( <code>cont</code> )                     |
| <code>M-u</code>       | up <code>arg</code> frames ( <code>up</code> )     |
| <code>M-d</code>       | down <code>arg</code> frames ( <code>down</code> ) |
| <code>C-x &amp;</code> | copy number from point, insert at end              |
| <code>C-x SPC</code>   | (in source file) set break at point                |

## GDB License

|                            |                                     |
|----------------------------|-------------------------------------|
| <code>show copying</code>  | Display GNU General Public License  |
| <code>show warranty</code> | There is NO WARRANTY for GDB.       |
|                            | Display full no-warranty statement. |

Copyright © 1991, 1992, 1993, 1998, 2000, 2010 Free Software Foundation, Inc. Author: Roland H. Pesch

The author assumes no responsibility for any errors on this card.

This card may be freely distributed under the terms of the GNU General Public License.

Please contribute to development of this card by annotating it. Improvements can be sent to bug-gdb@gnu.org.

GDB itself is free software; you are welcome to distribute copies of it under the terms of the GNU General Public License. There is absolutely no warranty for GDB.

# Q & A

---