CS35L-5

Week 8 Lec 2

GCC Flags

- `-fPIC`: Compiler directive to output position independent code, a characteristic required by shared libraries.
- `-lXXX`: Link with "`libXXX.so`"
  - Without `-L` to directly specify the path, `/usr/lib` is used.
- `-Ldir`: At **compile** time, find library from this path.
- `-Wl,rpath=.`: `-Wl` passes options to linker. `-rpath` at **runtime** finds `.so` from this path.
- `-c`: Generate object code from c code.
- `-shared`: Produce a shared object which can then be linked with other objects to form an executable.
- https://gcc.gnu.org/onlinedocs/gcc/Link-Options.html#Link-Options

Creating static and shared libs in GCC

- mymath.h

```
#ifndef _ MY_MATH_H
#define _ MY_MATH_H
void mul5(int *i);
void add1(int *i);
#endif
```

- mul5.c

```
#include "mymath.h"
void mul5(int *i)
{
  *i *= 5;
}
```

- add1.c

```
#include "mymath.h"
void add1(int *i)
{
  *i += 1;
}
```

- gcc -c mul5.c -o mul5.o
- gcc -c add1.c -o add1.o
- ar -cvq libmymath.**a** mul5.o add1.o ----> (static lib)
- gcc **-shared** -fpic -o libmymath.**so** mul5.o add1.o ------> (shared lib)

Dynamic loading

```
#include <stdio.h>
#include <dlfcn.h>

int main(int argc, char* argv[]) {
 int i = 10;
 void (*myfunc)(int *); void *dl_handle;
 char *error;
 dl_handle = dlopen("libmymath.so", RTLD_LAZY);//RTLD_NOW
 if(!dl_handle) {
  printf("dlopen() error - %s\n", dlerror()); return 1;
 }
 //Calling mul5(&i);
 myfunc = dlsym(dl_handle, "mul5"); error = dlerror();
 if(error != NULL) {
  printf("dlsym mul5 error - %s\n", error); return 1;
 }
 myfunc(&i);
 //Calling add1(&i);
 myfunc = dlsym(dl_handle, "add1"); error = dlerror();
 if(error != NULL) {
  printf("dlsym add1 error - %s\n", error); return 1;
 }
 myfunc(&i);
 printf("i = %d\n", i);
 dlclose(dl_handle);
 return 0;
}
```

- Copy the code into main.c
- gcc main.c -o main -ldl
- You will have to set the environment variable LD_LIBRARY_PATH to include the path that contains libmymath.so

Attributes of Functions

- Used to declare certain things about functions called in your program
  - Help the compiler optimize calls and check code
- Also used to control memory placement, code generation options or call/return conventions within the function being annotated
- Introduced by the **attribute** keyword on a declaration, followed by an attribute specification inside double parentheses

Attributes of Functions

- `__attribute__ ((__constructor__))`
  - Is run when `dlopen()` is called
- `__attribute__ ((__destructor__))`
  - Is run when `dlclose()` is called
- Example:

```
__attribute__ ((__constructor__))
void to_run_before (void) {
    printf("pre_func\n");
}
```

Homework 8

- Split `randall.c` into 4 separate files
- Stitch the files together via static and dynamic linking to create the program
- `randmain.c` must use *dynamic loading, dynamic linking* to link up with `randlibhw.c` and `randlibsw.c` (using `randlib.h`)
- Write the `randmain.mk` makefile to do the linking

Homework 8

- randall.c outputs N random bytes of data
  - Look at the code and understand it
    - Helper functions that check if hardware random number generator is available, and if it is, generates number
      - Hw RNG exists if RDRAND instruction exists
      - Uses cpuid to check whether CPU supports RDRAND (30th bit of ECX register is set)
    - Helper functions to generate random numbers using software implementation (/dev/urandom)
    - main function
      - Checks number of arguments (name of program, N)
      - Converts N to long integer, prints error message otherwise
      - Uses helper functions to generate random number using hw/sw

Homework 8

- Divide randall.c into dynamically linked modules and a main program
  - Don't want resulting executable to load code that it doesn't need (dynamic loading)
    - **randcpuid.c**: contains code that determines whether the current CPU has the RDRAND instruction. Should include randcpuid.h and include interface described by it.
    - **randlibhw.c**: contains the hardware implementation of the random number generator. Should include randlib.h and implement the interface described by it.
    - **randlibsw.c**: contains the software implementation of the random number generator. Should include randlib.h and implement the interface described by it.
    - **randmain.c**: contains the main program that glues together everything else. Should include randcpuid.h (as the corresponding module should be linked statically) but not randlib.h (as the corresponding module should be linked after main starts up). Depending on whether the hardware supports the RDRAND instruction, this main program should dynamically load the hardware-oriented or software-oriented implementation of randlib.

Change Management

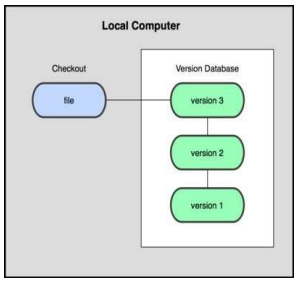
Software development process

- Involves making a lot of changes to code
  - New features added
  - Bugs fixed
  - Performance enhancements
- Software team has many people working on the same/different parts of code
- Many versions of software released
  - Ubuntu 10, Ubuntu 12, etc
  - Need to be able to fix bugs for Ubuntu 10 for customers using it, even though you have shipped Ubuntu 12.

Source/Version Control

- Track changes to code and other files related to the software
  - What new files were added?
  - What changes made to files?
  - Which version had what changes?
  - Which user made the changes?
- Track entire history of the software
- Version control software
  - GIT, Subversion, Perforce

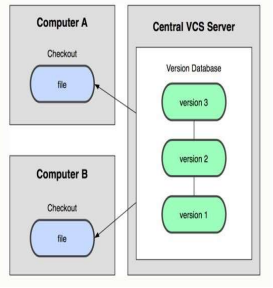
Local VCS



- Organize different versions as folders on the local machine
- No server involved
- Other users should copy it via disk/network

Image Source: git-scm.com

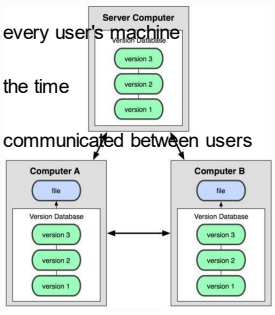
Centralized VCS



- Version history sits on a central server
- Users will get a working copy of the files
- Changes have to be committed to the server
- All users can get the changes

Image Source: git-scm.com

Distributed VCS

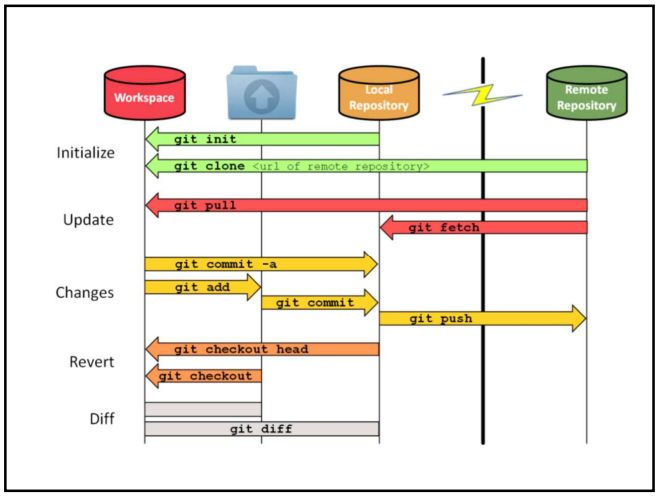


- Version history is replicated at
- Users have version control all
- Changes can be
- Git is distributed

Image Source: git-scm.com

Terms used

- **Repository**
  - Files and folder related to the software code
  - Full History of the software
- **Working copy**
  - Copy of software's files in the repository
- **Check-out**
  - To create a working copy of the repository
- **Check-in / Commit**
  - Write the changes made in the working copy to the repository
  - Commits are recorded by the VCS


Workspace
Local Repository
Remote Repository
Initialize
git init
git clone <url of remote repository>
Update
git pull
git fetch
Changes
git commit -a
git add
git commit
git push
Revert
git checkout head
git checkout
Diff
git diff