**Korn – Open Source Development Tools**

Debuggers

* Let’s you examine internal workings of your code as it runs
* Advantage
  + Step through code as it runs, without modifying code
  + Examine entire state of program
  + Modify values in the running program
  + View the state of a crash using core files
* Tools
  + Breakpoints
  + Recompile with proper debugging options
  + Use -g (e.g. gcc -g -c foo.c)
* Methods
  + Run debugger on your program, executing program from withing debugger to see what happens
  + Post-mortem mode: program has crashed; core dump
    - Stack trace
    - Chain of function calls that cause problem
    - Does not always pinpoint problem
* GDB, GNU debugger
  + gdb [<programfile> [<corefile> | PID]]
  + programfile = executable file
  + corefile = core dump of program
  + PID = id of process that’s alr running
  + Compile with -g for debug info
  + Basic GDB commands
    - File [<file>] – select file to debug
    - Run [<args>] – runs selected programs with arguments
    - Attach [<pid>] – attach gdb to a running process, pid
    - Kill – kills the process being debugged
    - Quit – quits the gdb program
    - Help [<topic>] – access the internal help documentation
    - c – continue, s – step one line entering called functions, n – next (step one line without entering functions), finish
  + Breakpoint tools
    - b [<where>] – breakpoint at where, can be in hex address, function name, etc.
    - r <expr> - watchpoint, which will break when expr is written to/read
    - info break – prints out a listing of all the breakpoints
    - clear [<where>] – clears a breakpoint at where
    - d [<nums>] – deletes breakpoint by number
  + Data manipulation tools
    - List [<where>] – prints out source code at where
    - Search [<regexp> - searches source code for regexp
    - Backtrace [<n>] – prints a backtrace of n levels deep
    - Info [<what>] – prints out info on what
    - p [<expr>] – prints out the evaluation of expr
    - set <name> <expr> - sets variables or arguments
    - return [<expr>] – returns expr from current function
    - jump <where> – jumps execution to where
  + Example of gdb at work:
    - gcc -g hello.c hello; gdb hello; list (program list); b 1 (breakpoint at line #1); c (continue to next breakpoint); quit
* System call tracing
  + strace (linux), truss (OpenSolaris)
  + dTrace – dynamic tracing framework
    - Instrumenting – inserting probes into kernel and user applications
    - Very slow, no disabled probe effect
    - Highly scriptable
    - Probes: provider:module:function:name
* Profilers
  + Learn where program spent its time
  + Measure frequency and duration of function call
  + Use gprof to profil c/c++ applications
    - Compile and link with -pg to enable profiling
    - Execute program to generate call graph profile data
    - Run gprof to analyze/display data

Make utility

* Maintaining and building computer programs
* Instructions stored in makefile
* Contains a build instruction for a project
* Only recompiles necessary files after a change – time saver
* Typical usage: executables are updated from object files that are in turn compiled from source files
* Format
  + <target>: <dependency list>; **TAB** <command>
  + Target – list of files that the command will generate
  + Dependency list – files and/or other targets that will be used to compile target
  + Must have a tab
  + First rule is the default <target> for make
* Make -f makefile; make target; make
* Make execute all commands associated with target if one of these is satisfied:
  + File target does not exist
  + File target exists but one of the source files in the dependency list has been modified more recently than target

tar: Tape Archiver

* General purpose archive utility
* tar [options] [file]
* tar acts on entire subtree, if file has subdirectories
* Options
  + -c, -f filename, -v (verbose), -x (extract), -z, -j, -t (table format), -r (append unconditionally), -u, -L, -m, -l

cpio: copying files

* copy file archives in from or out of tape or disk or to another location on the local machine
* Similar to tar
* Extract: cpio -idtu; **TAB** [patterns]
* Create: cpio -ov
* Pass-thru: cpio -pl directory

pax: replacement for cpio and tar

* Portable Archive eXchange format
* Part of POSIX

Packaging source: autoconf

RPM Package Manager

Debian Package Manager

* Doxygen – documentation generator
* Outpurs in html, pdf, etc.

IDE

* Code editing, compiling, debugging, versioning

**Korn - Revision Control Systems**

Revision control

* Version control
* Manage changes to soure code
* Multiple developers can work on code at the same time
* Changes to the same file are merged

Centralized Revision Control

* Central repository
* Client-Server
* Lock-Modify-Unlock
* Copy-Modify-Merge

Distributed Revision Control

* No center repository
* Distributed over internet
* Work directly from local repository
* Code from different repositories are merged later
* Work offline, fast operations
* Git, Mercurial, Bazaar

Concepts:

* Commit/Checkin – write/merge changes in working copy to repository
* Checkout – create local working copy of repository
* Trunk/mainline – base of the project
* Branch – separate copies of code under parallel independent development
* Tag/label – snapshot of a project
* Head – most recent commit/revision
* Merge – reconciling multiple changes to the same file
* Delta compression – record only differences in each revision, similar to diff

Basic Operations

* Setup directory – mkdir RCS
* Check in a new file – ci filename
* Checkout a file for reading – co filename
* Checkout a file for writing – co -l filename (acquires lock)
* Compare local copy to repository – rcsdiff [-r<ID>] filename

CVS

Subversion

* Better than CVS
* Directories, renames, meta-data are versioned
* Branching and tagging implemented as “copy” -no “branch-point tagging”
* Can use database backend
* Connect using svn, svn + ssh, http
* svnadmin create – create repository
* svn import – add local file
* svn commit is atomic
  + each commit creates a revision
  + Revision N is a state of repository after Nth commit
  + Updates and commit are separate
* Branching – svn copy, svn checkout
* Merging – svn merge, svn commit
* Tags and branches are the same

Git

* Import – tar, cd, git init
* Create first version – git add ., git commit
* Make changes – cloning and merging

**Nelson – C/C++ program development**

Converting C source to an executable

* Removes comments and creates intermediate i. file
* Compiles i. file and creates assemble file
* Raw code assembled by backend of gcc
* Assembly files converted into object o. files
* gcc invokes linker to create executable by merging .o file with libraries

Useful gcc options

* -E = only run the preprocessor, sending output to stdout
* -S = compile and create an assembly language .s file
* -c = Supress linkage, create object .o files for each input file
* -I dir = add the dir to the list of directories which will be searched for the include file
* -lname = search the name library when linking

Keeping programs up to date

* Make command

Building a library

* More convenient to group large numbers of object files in a library
* Command ar can manipulate an archive as a library
  + -r = adds an object file to the archive
  + -q = appends a file to the end of the archive
  + -x = extract
  + -d = deletes
  + -t = display table of contents
  + -v = verbose output