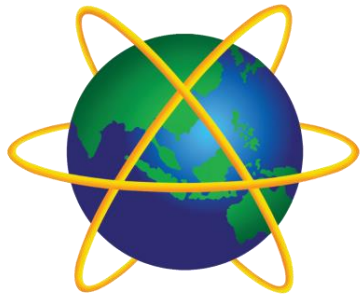


# System and Network Administration



**A · P · U**  
ASIA PACIFIC UNIVERSITY  
OF TECHNOLOGY & INNOVATION

gcc & make



# Source and Packages

---

- A Linux **distribution** is a collection of **utilities** bundled around the Linux **kernel**.
- **Source code** is the program in text file format, usually written in the language C
- A **binary file** is the result of **compiled** source code.
- A **dependency** is a component of the system that must already be installed before another program will function.
  - Some, but not all, compilation scripts will attempt a **dependency check** prior to installation.
- **Packages** are pre-configured binary files for specific distributions.

# Source and Packages

---

**Package managers** keep track of which packages have been installed, and perform a dependency check when you install software

Some distributions offer a tracking service that will notify you when new versions of installed packages are available.

Others automatically download packages from an official repository

## Debian/Ubuntu

Local – dpkg

Auto – apt-get

## RedHat

Local – rpm

Auto – yum

## Slackware

Local – pkgtool

Auto – *(none)*

***Celebrate  
Diversity!***

**TinyNet:** mount SlaxArchive CD

# Shared Libraries (.so) Dynamic Link Libraries (.dll)

---



- Loaded into RAM on demand
- Managed by some kernel routines which use an “index” to locate a required module
  - Special command used to do this – **ldconfig**
- **Must have the right libraries on your system**
  - dependencies** – missing or wrong version, cannot start
  - packages** – bundle libraries, but often depend on others
  - package managers** – help sort out dependencies

*It is very likely that you will actually need to configure these, for one application or another*

# Compiling: gcc & make



- There are four essential ingredients in a Linux distribution: the kernel code, the C compiler, C libraries, and the binutils package (the linker and other build tools).
- While there are exceptions, because some applications depend on certain features in the C libraries or the kernel (newer versions of the monkey webserver, for example), in general any code you can compile will work.
- Binaries other versions and distributions are highly likely to work, when the versions of their major components are similar.
- distrowatch.com tracks the characteristics of about a zillion distributions, but there are only 3 major ones to keep an eye on: Slackware, Ubuntu (Debian), and Redhat (CentOS).

# make, cmake, automake



- In these modern times we usually get software from a repository, but sometimes you need to compile it for yourself.
- make (or rather a Makefile) is a **buildsystem** - it drives the compiler and other build tools to build your code.
- If you intend your project to be multi-platform or widely usable, you really want a **buildsystem generator**
- CMake (cross-platform make) can produce Makefiles, Ninja build files, KDevelop or XCode projects, and Visual Studio solutions from the same starting point, the CMakeLists.txt file
- GNU Autotools (automake) integrate very well with building Linux distributions. They are not a general build system generator - they implement the GNU coding standards and nothing else.

- Make, Cmake, and GNU Autotools are actually insanely complicated, but they make the build process practically painless.
- Essentially, each application will have a **makefile** with various **targets** – some common ones are
  - make
  - make install
  - make clean
- There may be a layer “above” the makefile: **Cmake** **or** **configure** (GNU Autotools).
- We can add another layer to automate even more: **SlaxBuild**

# The basic process is:



- Use mc to copy the top directory for the source code from the archive file into /opt
- Switch to /opt, move down one level to the source code
  - Look for configure – this one uses automake
  - Look for CMakeLists.txt – this one uses Cmake



# The basic process is:



- IF you have a package that uses Automake or Cmake, THEN
  - Copy Template.SlaxBuild to another directory, rename it app.SlaxBuild, and copy it back to the same directory as the source code archive.
  - Follow the instructions there to make the necessary edits and customisations
  - Run ./app.SlaxBuild. If everything works, it will create two new files and install the application

# The basic process is:

---



- IF you do not have a package that uses Automake or Cmake, THEN
  - Check the Makefile for options you can customise
  - Check the Makefile for an INSTALL target and CLEAN
  - Check the Makefile for a CLEAN
- Run `make`. If everything works, it will create new files in the source directories
- If you can, run `make install` to move the new files from the source directories into their proper places in the filesystem
- If something goes wrong, run `make clean` to remove new files from the source directories, debug, and run `make` again

Study the SlaxBuild to see what it is doing

See TinyNet Images: ReadMe on the website to get a gcc VM set up

