

**Make made Easy**  
**or**  
**using xmkmf & Imake in an X environment**  
**with a diversion into tar**

*N.J.Gunton 10/98*

<b>Grouping</b>	Individual
<b>Prerequisites</b>	Access to a Unix System. Ability to use ftp.
<b>Courses</b>	UQC603S1, anyone interested.
<b>Requirements</b>	Command-line interface to Unix.
<b>Time</b>	½ to 1 hour.
<b>Summary</b>	Provides an introduction to the use of the Xmkmf, or X <b>make</b> <b>makefile</b> utility and the automatic generation of make files using imakefiles. Demonstrating the use of system and environment variables to control compilation.
<b>Objectives</b>	To understand how modern(?) software is built from source in an X-windows environment

## 1. The Background

Make is a utility, found on many systems, used to determine automatically which pieces of a large program need to be recompiled and to issue the commands necessary to recompile them. In practice `make` can be used for many purposes. It reads a file known as a `makefile` which describes the relationships between the program components, the commands needed to update them and often system specific information. For example the `makefile` below is used to compile an application called `bases`. It makes an assumption that the compiler to be used is the GNU C Compiler, `gcc`.

```
bases : main.o ourfiles.o listhandler.o
      gcc -g -o bases main.o ourfiles.o listhandler.o

main.o : main.c mydefs.h
      gcc -c -g main.c

ourfiles.o : ourfiles.c mydefs.h
      gcc -c -g ourfiles.c

listhandler.o : listhandler.c mydefs.h
      gcc -c -g listhandler.c

.PHONY : clean
clean :
      -rm bases main.o ourfiles.o listhandler.o
```

It is not uncommon in the UNIX world for applications or other software to be distributed as the source-code and to be compiled for a specific platform by the System Administrator. This can require editing of the `makefile` to successfully build the software. For applications that run in the X-window system this process can be automated.

## imake and the Imakefile

`imake` is a preprocessor interface to the `make` utility and is used to automate the production of `makefiles` by the use of a template. This allows the system dependent features such as command names or flags, special rules and compiler options to be kept separate from the descriptions of the software components to be built. `Imake` uses a whole set of files in order to do its job. See the relevant `man` page for full details. In practice it is more usual to use `xmkmf` to generate the `makefile` from the `Imakefile` when building X applications. All the arguments that are required by `imake` are configured into `xmkmf` when the X system is initially built for a particular platform. Again look at the `man` page if you're interested.

## 2. What to do....The Overview

- First create a new directory within your home directory called `src` as a repository for source code†.
- Then obtain the source code for an application called XPostitPlus-2.3. It can be found as an archive file in `~ngunton/Public/XPostitPlus-2.3.tar.gz`. Copy it into your new directory. Or ftp it from `ftp.zcu.cz/ftp/mirrors/X-11-contrib/office`.
- Make sure that your new directory is your current working directory (how do you do this?) and extract the files that you need from the archive file. See the discussion below for details on how to do this.
- Check the files for any relevant documentation. In other words read the `README` file. There is almost always one and it will provide general information at the very least and detailed instructions on building the software if you are lucky.. The other important file is `FileList`, if it exists. This file will list what should be included in the distribution.

Testing your UNIX knowledge....How can you read the `README` and `FileList` files?

## 3. What to do.....The Details

### A diversion into tar & gzip

- You will often need to unpack files that you have obtained. Unix files are commonly distributed as `.tgz` or `.tar.gz` files. These are known as 'tarred and zipped' files. Tar is an archiving utility which packs many files into a single file. It will also preserve the directory structure of the archived files. Its name comes from **t**ape **a**rchive as its original use was(is) to make tape backups of systems. A set of files or directories and files that have been archived in this way is known as a tar file. Once a tar file has been made it is often then compressed, either with the unix utility `compress` or more commonly with `gzip`. `compress` has been around longer and is slightly less efficient at compressing a file. `compress` will save compressed files as `filename.Z` while `gzip` will create `filename.gz`. If you are lucky you will have the GNU version of `tar` which can handle the (un)zipping of the file as well as the archiving of the files. Unfortunately at UWE you will have to do it in 2 stages. The table below shows typical use of `g(un)zip`.

**Table 1:** The gzip command

<b>command</b>	<b>filename</b>	<b>becomes</b>
<code>gzip</code>	<code>filename</code>	<code>filename.gz</code>
<code>gzip</code>	<code>filename.tar</code>	<code>filename.tar.gz</code>
<code>gunzip</code>	<code>filename.gz</code>	<code>filename</code>
<code>gunzip</code>	<code>filename.tar.gz</code>	<code>filename.tar</code>

† You might choose to put it in a directory such as `\usr\local\src` or `\common\public\src` if you were the systems administrator.

- When you have unzipped the `XpostitPlus-2.3.tar.gz` archive you should be left with `XpostitPlus-2.3.tar`. Check your directory contents to confirm this. The next phase is to turn the archive file back into the original set of files (and possibly sub-directories). It is possible to check the contents of an archive file without untarring it. It is also possible to extract specific files from an archive. The table below provides a useful subset of `tar` options. These options can be combined although some are mutually incompatible. (eg *combining the creation and extraction of an archive.*) The general principle is

`tar functionoptions files...`

where *function* is a single letter defining the function to perform and *options* is a list of (single letter) options to the function. *files...* is the list of files to be packed or unpacked.

**Table 2: Common `tar` functions**

<code>c</code>	create a new archive
<code>x</code>	extract files from an archive
<code>t</code>	list the contents of an archive
<code>r</code>	append files to the end of an archive
<code>u</code>	update files that are newer than archive
<code>d</code>	compare archive files with those in filesystem
<code>z‡</code>	(un)zip (before)after (un)tarring

Only one of these functions can be selected at a time, with the sole exception of `z` which may be combined with `c` or `x`. The first three in Table 2 are the most commonly used functions. There are many options to `tar` of which the most frequently used are given in Table 3.

- Given the information in the tables what will be the result of executing the following statements?
  - `tar tvf XPostitPlus-2.3.tar`
  - `gunzip mycode.c.gz`
  - `tar cvvf mydir.tar mydir`
  - `gunzip myfiles.tar.gz | tar xvf -`
  - `tar xvvf XPostitPlus-2.3.tar XPostitPlus-2.3/README`

**Table 3: Common `tar` options**

<code>v</code>	print verbose information. Can use multiple <code>v</code> 's
<code>k</code>	keep any existing files when extracting files ie. don't overwrite files of the same name.
<code>f filename</code>	specify <i>filename</i> as the tar file to be read or written

- This should give you sufficient to find out which files are in the archive file `XPostitPlus-2.3.tar` and to unpack the archive into your subdirectory.

---

‡ only with the GNU version of `tar`.

## Unpacking & Building the Code

Start by uncompressing the file `XPostitPlus-2.3.tar.gz` with the `gunzip` command. This should leave you with the file `XPostitPlus-2.3.tar`. Rather than just unpacking the code in one go, get a file listing of the contents. This can be written direct to standard output or, perhaps more usefully, saved in a file. This can be done as follows to get a contents list to the screen.

```
tar tvf XPostitPlus-2.3.tar | more
```

or

```
tar tvvf XPostitPlus-2.3.tar | more
```

To create a file with very verbose information execute

```
tar tvvf XPostitPlus-2.3.tar > XPostitPlus-2.3.contents
```

Study the file list to get an idea of the structure of the archive. You should be able to identify several subdirectories, the various source files and header files (`.h` files). There are also bitmap files, manual pages, history or revision files and readme's.

### Do the following

- Extract the file `README` and read it.
- Extract the file `FileList` and compare it to the contents of the archive
- Extract the file `Imakefile`. This is where you might have to make changes in order to be able to compile the application. Not all applications would require changes to be made, one common way of finding out is to read the documentation, run `xmkmf` and see what happens. Is this a good idea?. Note that many of the lines begin with the phrase `XCOMM`. This is a comment line which `imake` will change to match that used by the local version of `make`. Notice also that the file carries a revision and version history at the beginning. This is a good example of brief but useful documentation as are the comments embedded within the file. It is also an extremely good idea to make a backup of this file before you start editing it and to comment any changes that you make with the date and the reason.

Once you have got a reasonable idea of what is going on, have checked the file list and have studied the sequence of instructions in the documentation, it is time to unpack the whole archive. If you have already unpacked the above files into a subdirectory then it might be useful to use the `k` option to prevent overwriting those files‡ Untar the whole archive and follow the instructions in the `README`. In brief these are

- Make any changes that are necessary to the `imake` file.
- Run `xmkmf` to create a `makefile` tailored specifically to your machine.
- Execute `make depend`. This runs the `make` command with the instructions found in the section of the `makefile` labelled `depend`. This sets up a header file needed by the main compilation.
- Run `make` on its own. By default it will look for a file called either `Makefile` or `makefile` and execute the instructions within it. It will build the application if all is well.
  - What happens? Are there any errors or warning messages? If there are any warnings can they be ignored?
  - If there are any errors, make any changes needed to the `imake` file and run `make clean`. This will tidy up any stale files left over from the previous attempt. Once this is done try running `make` again. Remember to comment any changes within the `imakefile`.
  - Repeat these steps until `XPostitPlus` builds successfully.‡‡ Explain to your tutor

---

‡ Although in this case it doesn't matter, it provides an opportunity to try out an option.

‡‡ In this case there is only one edit needed to `imakefile`.

any edits you have made and what you think the impact will be.

#### 4. Testing and Installation

When you obtain software in this fashion it often comes with associated `man` pages. These are sometimes in a compressed form, in which case you'll need to unzip them, either way they are often not quite plain text files. They are `troff` files and contain formatting information. The formatting instructions are in plain text but it's like reading raw HTML only worse. The `man` utility knows how to handle them. There may also be plain text versions. In either case read the `man` pages and test the functionality of your built product.

As a systems administrator you may well get asked by users to install a particular piece of software that they have found so think carefully about the testing process. If you are familiar with C then have a look at the source code as well. When testing is complete the executable would normally be put in either `/usr/local/bin` or `/usr/bin` or `/common/public/bin` depending on how the administrator chose to structure the system.

#### Discussion Points

Discuss the following points with your colleagues and your lab tutor, find examples where appropriate. Find a small X application, download it and build it.

- Sometimes you find that the source code for a package includes patches or patch files. What are they and what do you do with them.
- You had to edit the `imakefile`. should you report this to the originators of the code? Under what circumstances might you send in a report. (hint.. read the `readme`).
- Can you trust the code that you have built and run. Might it do something terrible. How can you be sure that the code you have obtained is the correct version, most up-to-date?