

MACTEX OVERVIEW

RICHARD KOCH

Part A: Binaries

1. INTRODUCTION AND HISTORY

This is the main document explaining how to construct MacTeX.

Wendy McKay conceived the idea of a Macintosh install package which would provide everything needed to use TeX on a Macintosh. After advocating for this package at a couple of earlier TUG conferences, she organized a lunch for Macintosh users at the TUG Practical TeX Meeting, 2005, in Chapel Hill, North Carolina. At that lunch, she pointed to Jonathan Kew and assigned the construction of the installer to him. Jonathan had to leave the conference the next morning, and we expected that he would construct the package after he returned to England. Instead, Jonathan stayed up all night and had a package the next morning. Over breakfast, he willed maintenance of it to me.

Originally, MacTeX installed a TeX Distribution created by Gerben Wierda, based on teTeX. But in 2006, Thomas Esser, the author of teTeX, announced that the project was ending and recommended that users switch to TeX Live. Gerben Wierda then began revising his distribution to contain a mixture of teTeX and TeX Live, announcing the new distribution, gwTeX, in November, 2006 at the annual TUG meeting in Marrakesh, Morocco. But at that same meeting, Gerben held up a sign containing the words “I quit” and announced that he would no longer support his distribution. After that I constructed three test versions of the install package, one installing the old teTeX, one installing the new gwTeX, and one installing the full TeX Live. To my delight, the TUG authorities pushed for the package based on the full TeX Live, and since 2007, that is what MacTeX installs.

2. BINARIES

This document is provided inside a build tree for MacTeX. Do not rearrange folders in this tree. As MacTeX is constructed, these folders will gradually be filled with bits and pieces of the package, until finally the complete package is inside one of the folders. Some folders contain a subfolder named MiscDocs. This is extra material that might be useful, but reading the material is probably not necessary.

MacTeX supports at least all versions of macOS for which Apple is currently providing security patches. These are the current system and the two previous systems. Thus in

Date: February 28, 2024.

2024 we should support Monterey, Ventura, and Sonoma. Around September, 2024, Apple will release the next operating system, and we always support that as well by working carefully with the beta release starting in June.

For the past several years, we have supported even more systems: Mojave, Catalina, Big Sur, Monterey, Ventura, and Sonoma. This includes all systems that ever ran on Arm, because the first Arm machines ran Big Sur. Unless something drastic happens, we will support the same systems in 2024.

Binaries must be compiled twice, once on an Arm machine (this year running Big Sur) and once on an Intel machine (this year running Mojave). They are then combined using lipo, signed, and zipped up to be sent to TeX Live. Once binaries are compiled, all remaining steps can be done on a more recent system. In 2024 all steps except compiling are done on an Arm system running Sonoma.

When MacTeX and TeX Live are updated for a new year, the very first step is to compile the TeX Live Macintosh binaries. All material needed for this step is contained in the folder named “Binary” in the build system. Inside this folder is a TextEdit document named BuildStatus-2024. This contains the full instructions for making TeX Live binaries. It contains actual commands which can be copied and pasted into Terminal.

On both the Intel machine and the Arm machine used to compile binaries, create an empty folder in the home directory named texlive2024dev. Then an rsync command in the document populates it with the current sources. This same command updates the directory when the sources change. The original step takes several minutes, but updating typically takes 30 seconds or less. Here is that command

```
cd
cd texlive2024dev
rsync -a --delete --exclude=.svn tug.org::tldevsrc/Build/source .
```

After the source is obtained, compiling is straightforward. But a few preliminary steps are required before building.

A small number of the binaries link with X11 libraries. They are pdfopen, pdfclose, xdvixaw, and xindy. Originally, Apple supported X11 on macOS with an install package named XQuartz. This package was only updated when new systems were introduced, and X11 users complained, so XQuartz became an independent open source project. The releases can be obtained at <https://www.xquartz.org>. TeX Live links a few programs with the X11 Library, so X11 must be installed on each build machine, but need not be running.

One program, xindy, requires Lisp. Before building binaries, it is necessary to obtain and compile Lisp. This is explained in the following section.

3. LISP

Lisp is not part of TeX Live, but it is required on build systems to create xindy, a program in TeX Live.

When xindy was added to the TeX Live toolchain, we were told to first compile clisp-2.49 and put it in a spot which could be called by the build system. This procedure worked.

Shortly afterward, we needed to support both PowerPC and Intel processors. Compiling xindy produces xindy and two support files named xindy.mem and xindy.run. It turned out that xindy.mem depended on the processor. The author of Xindy had to modify his code so on a Macintosh the correct version of xindy.mem would be used.

Later new versions CLisp were introduced, but did not work on some of our systems, so we continued using clisp-2.49. When Apple made the Arm transition, clisp-2.49 didn't compile on Arm, so for a couple of years we only produced Intel code, and Arm users needed to run it with Rosetta.

Then I received an email from Roberto Avanzi, roberto.avanzi@gmail.com. This email explained how to compile the modern version of CLisp on both Arm and Intel Macintosh machines. The last section of the document BuildStatus-2024 lists the url where CLisp source can be obtained, and the Avanzi instructions to compile it. Lisp is installed in the home directory of each system used to compile TeX Live, and compiling is very straightforward. This step must be done just once each year. Indeed when the same computer is used to compile TeX Live in successive years, the previous CLisp still works fine.

4. COMPILING TeX LIVE

First go to /usr/local and temporarily rename the bin, lib, include, share, and texlive folders so the system will not accidentally use some of this code.

Now you are ready to compile. On both the Intel and the Arm machine, execute

```
cd
cd texlive2024dev/source
export STRIP="strip -u -r"
./Build --enable-xindy CLISP=/Users/koch/CLisp/bin/clisp
```

If there is an error, it must be reported to TeX Live and an unpleasant debugging session will commence. But if there is no error and the build passes all tests automatically done at the end, there will be a folder in texlive2024dev/source/inst/bin with a name like aarch64-apple-darwin23.2.0 on an Arm machine, or x86_64-apple-darwin18.7.0 on an Intel machine. The numbers in these names may be different in other years. Move the entire folder from both the Arm and Intel machines to MacTeX-2024/Binary/RawCode2024 on the machine used for the final construction of MacTeX. (I use a small portable transfer disk with a

folder RawCode2024. I empty this folder before building, and move both Arm and Intel builds to this folder on the transfer disk and via it to the Sonoma machine.)

5. ASYMPTOTE

One other binary requires special attention, asy or Asymptote. It is not compiled automatically by the build system used in the previous section.

There is a folder named Asymptote in the main TeX Live build folder. Create a copy of this folder in your home directory. The folder contains an instruction sheet, “AsymptoteBuild-2024”, and a Sample folder to test the final result. Follow the instructions in the instruction sheet. They explain how to download and compile four libraries used by asymptote. This needs to be done only once a year. The libraries will be placed in the Asymptote Folder and thus will not modify /usr/local. The asymptote source itself is in the TeX Live source and obtained when you obtain that source. “AsymptoteBuild-2024” explains how to compile it. On Arm, rename the final binary “asy-Arm” and on Intel name it “asy-Intel”. Move these binaries to the machine used to create MacTeX and its folder MacTeX-2024/Binary/RawCode2024.

6. PACKAGING UNIVERSAL-DARWIN

All remaining steps are done on the main computer, so in 2024 on an Arm machine running Sonoma. After compiling on an Intel and an Arm machine, and moving the resulting files to MacTeX-2024/Binary/RawCode2024 on Sonoma, first empty the folder Binary/RawCode, and then copy the contents of RawCode2024 to RawCode. This guarantees that if a mistake is made in the following steps, it is not necessary to return to the older machines and rebuild again.

The goal of the next steps is to lipo the binaries to form universal binaries, sign these binaries, and package them into a single zip file universal-darwin.tar.xz, which is sent to Karl Berry to add to TeX Live. This will involve using some scripts in the MacTeX build system. The key scripts are named

```
lipounsignall.sh
lipocreateuniversaldarwin.sh
liposignuniversaldarwin.sh
liporebuild.sh
```

All of these scripts except liposignuniversaldarwin.sh contain the following two lines near the top:

```
ArmFiles=aarch64-apple-darwin23.2.0
IntelFiles=x86_64-apple-darwin18.7.0
```

Once each year, these scripts must be edited to replace these lines with the names created when compiling on the particular versions of macOS used to make Arm and Intel binaries.

The scripts call the compression utility `xz`, which is not part of macOS. Therefore, the Binary folder must contain this binary.

To create the initial `universal-darwin.tar.xz` sent to TeX Live, run three of the four scripts in the following order:

```
sh lipounsignall.sh
sh lipocreateuniversaldarwin.sh
sh liposignuniversaldarwin.sh
```

The script `liporebuild.sh` will not be used in this step.

What do these scripts do? The first script removes all developer signatures from the binaries in Raw Code. This is necessary because some versions of XCode, particularly on Arm, sign binaries automatically even when they are compiled on the command line. Experiments suggest that these signatures interfere with the `lipo` command.

The script `lipocreateuniversaldarwin` removes the `universal-darwin` directory and its compressed file, if they exist, and then creates a new empty `universal-darwin` directory. Then it looks at all files in the Intel subdirectory of RawCode. If a file is not a symbolic link and is instead an executable, and if there is a corresponding Arm executable in the Arm directory, then the script creates a universal binary from these pieces and places it in `universal-darwin`. The script also copies Intel symbolic links and scripts directly into `universal-darwin`. Then it creates a universal binary from `asy-Intel` and `asy-Arm` and places it in `universal-darwin`. It creates a symbolic link `xasy` in `universal-darwin` related to `asy`. Finally, it renames `xindy.mem`, which was copied to `universal-darwin` from the Intel directory in RawCode, to `xindy-x86_64.mem`, and copies the corresponding Arm file in RawCode to `xindy-arm64.mem`.

After these steps, `universal-darwin` contains unsigned copies of the complete macOS TeX Live binaries.

The script `liposignuniversaldarwin` signs all of the binaries in `universal-darwin`, and adopts a hardened runtime for these binaries. A small number of binaries require exceptions. These binaries are signed again, this time with a hardened runtime allowing the required exceptions.

Then `liposignuniversaldarwin` creates `universal-darwin.tar`, and compresses it to `universal-darwin.tar.xz`. This final compression step takes a little time, because `xz` takes longer to compress but decompresses very rapidly.

The file `universal-darwin.tar.xz` should be sent to Karl Berry for inclusion in the initial TeX Live 2024 pretest.

7. MINOR DETAILS OF THE PREVIOUS PROCESS

Since the Macintosh does not come with a copy of xz, it is necessary to compile it and place a copy of the binary in the Binary-2023 directory so the script can find it. We'll describe compiling xz in the next section.

Open the `lipocreateuniversaldarwin` script in an editor. You'll notice is that it uses the "file" command to determine whether a file is a link, a script, an Intel-only binary, or a universal binary. Only the last two types of files are signed.

Next notice that for Intel-only binaries like `xindy`, `file(xindy)` outputs "xindy: Mach-O 64-bit executable x86_64". But for universal binaries like `tex`, `file(tex)` outputs a longer string which only starts with the script "tex: Mach-O universal binary" and then continues with text which we ignore.

These files are signed using the command

```
codesign -s "Developer ID Application: Richard Koch"
```

Another person running this script would have to pay Apple \$100 a year to become an Apple Developer. They would then obtain two licenses called "Developer ID Application" and "Developer ID Installer". The first license is used for applications, and the second will be used for MacTeX and other install packages. When these licenses are obtained, information about them is stored in the computer's Keychain. This information is accessed to sign the code. Therefore as written, the script will only work for me, and another person will need to change to their own name and insure that the keychain has appropriate information from Apple. Incidentally, there is a fancy way to move developer keychain information from one computer to another. Thus if I buy and use a new machine, the scripts won't work until I move my keychain developer information using the fancy method.

Notice that the signing commands have a flag

```
--options=runtime
```

This flag causes the application to adopt Apple's "hardened runtime." Such an application is not allowed to perform certain dangerous tasks like accessing the camera, or the user's location, or the user's address book. The application also cannot link with a third party library, or execute JIT-compiled code. A full list of prohibited actions can be found in the document "Hardened Runtime Entitlements" in the Binary/Hardened-Runtimes directory.

However, for each prohibition, the application can file an "exception." These exceptions are always automatically granted. So the idea is that Apple doesn't restrict programs in any way, but if a programmer doesn't intend to use a dangerous feature, they the programmer can make sure that even illegal code inserted by an attacker cannot use that feature.

After signing all binaries, the script `lipoharden` considers a few applications that require exceptions. These are signed again, and this time an additional flag "-entitlements" lists

a file containing the required entitlements for that program. Three binaries (mf, xdv-xaw, and dvisvgm), link with an X11 library which is created by a third party, so they need an entitlement for that purpose. Four applications related to luatex require an entitlement to run JIT code.

8. xz

Certain portions of the TeX Live Unix Install Script, and the tlmgr code in TeX Live, use wget and curl to download code, and use xz to compress and decompress files. The Mac already has curl, so nothing has to be done about it. Since curl suffices, we do not provide wget. But early each year, before seriously working on the binaries, we must compile xz and send the resulting universal code to Karl. The folder xz-code in the Binary directory contains all scripts and tools to do this. We only use limited xz abilities, so the configure command disables many features.

In 2024 the source was in <https://xz.tukaani.org/xz-utils/> and we switched to xz-5.6.0. The instructions below otherwise still work. Use otool -L xz-Arm to see if there are shared libraries. None should exist except /usr/lib/libSystem. Use lipo -archs xz to see which architectures are supported.

To compile, search the internet for source code; we currently use xz-5.2.5. Uncompress the tar file and compile xz using the instructions

```
cd xz-5.2.5
./configure --disable-nls --disable-shared --disable-threads
make
strip src/xz/xz
cp src/xz/xz ../xz.Arm
```

On Intel, replace the last xz.Arm with xz.Intel. Place these two binaries in the RawCode directory. Run

```
sh lipoxz.sh
```

to produce xz-universal-darwin and xz-universal-darwin.tar.xz.

9. LZ4

Certain portions of tlmgr also use lz4, and Karl may ask for binaries for this program. See lz4-code in the build system. To compile, search the internet for source code; we currently use lz4-9.3. Uncompress the tar file and compile lz4 using the instructions

```
cd lz4-1.9.3
make
strip lz4
cp lz4 ../lz4.Arm
```

On Intel, replace the last lz4.Arm with lz4.Intel. Place these two binaries in the RawCode directory. Run

```
sh lipolz4.sh
```

to produce lz4-universal-darwin and lz4-universal-darwin.tar.xz.

10. UPDATING BINARIES

It will be necessary to update the TeX Live binaries several times during the pretest period as new code is installed and bugs are fixed. To do that, we recreate the full set of binaries as described earlier, and send this full set to Karl. But when he installs the set in the pretest, he only installs files which have been changed, so testers don't need to reinstall the full binary set. Unhappily, the lipo stage of our process changes the state of files and ruins this plan.

A modified process fixes this problem. When Karl asks for a rebuild, first rename the RawCode folder to RawCodeOld. If there is already a RawCodeOld folder, rename it RawCodeJan26 (or another date) and put it in RawCodeHistory.

Next uncompress the file universal-darwin.tar.xz in RawCodeOld.

Then compile and add files to RawCode as usual. But in the final stage, instead of running the three lipo files listed earlier, run

```
sh lipounsignall.sh  
sh liporebuild.sh
```

The liporebuild script compares all Intel files in RawCode to the corresponding Intel files in RawCodeOld. If the two files are the same, then it copies the corresponding file from RawCodeOld/universal-darwin to the new universal-darwin folder begin constructed. Otherwise it creates a new file by using lipo as usual, and signs this file if necessary. Details can be seen by reading liporebuild. The end result is that if a file did not change, then the old file from the previous universal-darwin is sent to Karl, but otherwise a new file is sent.

The script also reports all changed files. This detail can be sent to Karl with the new universal-darwin, so he can check that the expected files were changed.

I hope this continues to work in the future, because liporebuild is the most complicated shell script I have ever written. (By Karl's standards, it was a triviality.)

11. LUAMETATEX

In the middle of the pretest for 2023, the ConTeXt team submitted a ConTeXt distribution with a single binary named `luametatex`. Unfortunately, this binary was created using `cmake`, which is not part of the TeX Live build system. Consequently, the ConTeXt team will now create their own binaries for all platforms and submit the binaries directly to TeX Live. This creates problems for MacTeX because those binaries would not be signed and thus could not be included in our install package.

I refused to install `cmake` on my systems because I maintain a plain vanilla system for the sake of TeXShop. But with important help of Mojca Miklavc, I discovered that it is easy to deal with `luametatex` and `cmake` without breaking this promise. The key point is that there is a CMake packaged as a macOS application. This application is self contained; it installs nothing in `/usr/local` or the XCode build system. The application can be pointed to the source code of a `cmake`-based project, and with a few simple steps it creates a make file for the platform on which it is running. After that, a simple “make” rapidly produces `luametatex`.

Here are the details. The application can be downloaded from <https://github.com/Kitware/CMake> and indeed the link <https://github.com/Kitware/CMake/releases/download/v3.25.2/cmake03.25.20macos-universal.dmg> produces a `dmg` file with the application. Place a copy of this program in the Application directory of all machines used to build `luametatex`.

The source code for `luametatex` was obtained from <https://github.com/texttextgarden/luametatex>. The ConTeXt people might provide a different location in a future year. This location ultimately produces a folder named `luametatex-main`. Put that folder anywhere in the home directory of the machine used to compile `luametatex`.

Open the Application CMake. A dialog will appear with a number of fields. The top line of the dialog reads

Where is the source code:

Next to that field is a button named “Browse Source...”. Click it to bring up a Finder window and select the folder `luametatex-main` in this window. This will fill in the top line.

The next line, titled Preset, can be ignored. The following line is titled

Where to build the binaries:

Next to that field is a button named “Browser Build...”. Click it to bring up a Finder window pointed to the contents of `luametatex-main`. Create a new folder in this folder named “inst”.

Near the bottom of the dialog is a button named "Configure". Push this, and notice that the terminal window at the bottom shows the resulting action. In the end, CMake will create some items in the middle portion named

```
CMAKE-BUILD_TYPE
CMAKE-INSTALL_PREFIX    /usr/local
CMAKE_OSX_ARCHITECTURES
CMAKE_OSX_DEPLOYMENT_TARGET
CMAKE_OSX_SYSROOT       /Applications/Xcode.app/Contents/Developer/Pi...
```

These items will appear in red so the user can edit them, but editing is not needed. For instance the second item determines where luametateX will be installed if we type "sudo make install" but we will never do that.

At the bottom of the dialog, push "Generate". This item will create some files in luametateX-main/inst. The most important is named "Makefile".

Using Terminal, change directory to luametateX-main/inst and type "make". The project will be built in the usual way and the binary luametateX will appear in luametateX-main/inst.

Move this to the folder RawCodeConTeXt the Binary-ConTeXt directory of the computer used for final construction of MacTeX-2024. Rename the two binaries luametateX-Arm and luametateX-Intel.

Change to the directory Binary-ConTeXt, and run the command

```
sh lipocontext.sh
```

to lipo the two files into a universal binary file, sign this file, and create a file "universal-darwin-context.tar.xz". Send this file to the ConTeXt people, who will include it in the material they send to TeX Live.

Part B: BasicTeX

12. INSTALL PACKAGES

A glance at the MacTeX-2024 directory reveals that most folders install one or another version of TeX, or a piece of the full MacTeX. All of these folders are constructed in the same way, so it suffices to describe the specifics in a single case. Let us concentrate on BasicTeX-2024.

The folder for this item contains a directory named “root”. This directory will eventually contain a duplicate of part of the Mac file tree, containing exactly the new material to be installed on that machine. For example, the root folder for BasicTeX-2024 will contain `root/usr/local/texlive/2024basic/...` and the string of dots will be the full contents of BasicTeX. The BasicTeX subfolder of MacTeX-2024 contains scripts which construct this root folder, and then scripts which create the install package from this root, and finally scripts which sign and notarize this package.

To create the root folder, we will install BasicTeX on our machine using the TeX Live Unix Install Script. This script is available at <https://tug.org/texlive/acquire-netinstall.html> by clicking the link titled “install-tl-unx.tar.gz”. The pretest of Tex Live contains a similar script. This script, when run, downloads a large number of packages over the internet and installs them on the local machine.

However, it is much more convenient to download the full set of install packages once and for all and keep them on our local machine, since many different installs will be done from the collection. We place this full set in the directory `~/texlive2024pretest`.

Even before the pretest is available, it is useful to remake everything using the previous year’s packages. The following command will populate `~/texlive2024pretest` from the CTAN repository in Utah. Here the two backslashes should be removed and the last two lines combined into one line.

```
cd
cd texlive2024pretest
rsync -a --delete --exclude="mactex*" \\  
rsync://ctan.math.utah.edu/CTAN/systems/texlive/tlnet .
```

When the pretest becomes available, the following command downloads the pretest directly from the Utah mirror server in the same way.

```
cd
cd texlive2024pretest
rsync -a --delete --exclude="mactex*" ftp.math.utah.edu::tlpretest .
```

It is also possible to download the pretest directly from TUG’s server, but this loads the server and should usually be avoided. Also, it may create an extra folder between `texlive2024pretest` and the contents downloaded, changing some later scripts.

```
cd
cd texlive2024pretest
rsync -a --delete --exclude="mactex*" ftp.tug.org::texlive/tlpretest .
```

Both are large downloads that initially take around an hour. But the same commands also update `~/texlive2024pretest` after changes are made, and these commands take only a couple of minutes.

13. BASIC_{TEX}-2024

We now turn to the BasicTeX-2024 folder. Essentially the same steps we describe here apply to other install folders as well.

To create BasicTeX-2024, we first install BasicTeX and then construct the “root” folder from this installation. Begin by deactivating key elements of `/usr/local`:

```
cd /usr/local
sudo mv bin bin-temp
sudo mv lib lib-temp
sudo mv share share-temp
sudo mv texlive texlive-temp
```

Next run the texlive install script in `~/texlive2024pretest`:

```
cd
cd texlive2024pretest (might be: cd texlive2024pretest/tlnet)
sudo ./install-tl
```

The following text will appear in Terminal:

```
<B> set binary platforms: 1 out of 16

<S> set installation scheme: scheme-full

<C> set installation collections:
    40 collections out of 41, disk space required: 7189 MB

<D> set directories:
    TEXDIR (the main TeX directory):
        /usr/local/texlive/2024
    TEXMFLOCAL (directory for site-wide local files):
        /usr/local/texlive/texmf-local
    TEXMFSYSVAR (directory for variable and automatically generated data):
        /usr/local/texlive/2024/texmf-var
    TEXMFSYSCONFIG (directory for local config):
        /usr/local/texlive/2024/texmf-config
    TEXMFVAR (personal directory for variable and automatically generated data):
        ~/Library/texlive/2024/texmf-var
    TEXMFCONFIG (personal directory for local config):
        ~/Library/texlive/2024/texmf-config
    TEXMFHOME (directory for user-specific files):
        ~/Library/texmf

<O> options:
    [ ] use letter size instead of A4 by default
    [X] allow execution of restricted list of programs via \write18
    [X] create all format files
    [X] install macro/font doc tree
    [X] install macro/font source tree
    [ ] create symlinks to standard directories

<V> set up for portable installation

Actions:
<I> start installation to hard disk
<P> save installation profile to 'texlive.profile' and exit
<Q> quit
```

Enter command:

We first change any default answers that are wrong for us. After that, the installation proceeds without further help. It downloads and installs a large number of packages over the network, or directly finds these packages in `texlive2024pretest`.

The TeX Live authors have been kind to the Macintosh, and most default answers are already correct. Let us look at the questions and answers. The install script will pick the binaries suitable for the machine being used. In 2024 it will pick `universal-darwin`. Using the first set of options, we can add additional binary sets to the installation, but we won't.

The second "S" choice picks an installation scheme. By default, everything is installed and that is what MacTeX will do. But for BasicTeX we select a smaller set of packages. So type S and push return and the display will change to the following:

Select scheme:

```
a [X] full scheme (everything)
b [ ] medium scheme (small + more packages and languages)
c [ ] small scheme (basic + xetex, metapost, a few languages)
d [ ] basic scheme (plain and latex)
e [ ] minimal scheme (plain only)
f [ ] ConTeXt scheme
g [ ] GUST TeX Live scheme
h [ ] infrastructure-only scheme (no TeX at all)
i [ ] teTeX scheme (more than medium, but nowhere near full)
j [ ] custom selection of collections
```

Actions: (disk space required: 7189 MB)

```
<R> return to main menu
<Q> quit
```

Enter letter to select scheme:

Type "c" to pick the small scheme, which gives Basic TeX. As a matter of fact, Basic TeX came first in the Macintosh world, and then TeX Live copied our choices to form this scheme. Then push "R" to return to the main choices.

The next question can be ignored. But we need to change some of the directory choices. Actually the TeX Live people have again been kind to the Macintosh and the default choices in our list are the correct ones for the full MacTeX.

I won't show the choice dialog, but will walk our way through it. The first directory is the location where TeX will be installed. MacTeX installs in a directory named 2024, but BasicTeX installs in a directory named 2024basic so both installations can coexist. So change the first option to `/usr/local/texlive/2024basic`. The script automatically changes several

other names. But we must change options 5 and 6 from “ /Library/texlive/2024/texmf-var” to “ /Library/texlive/2024basic/texmf-var” and from “ /Library/texlive/2024/texmf-config” to “ /Library/texlive/2024basic/texmf-config”.

Next we change the ”options”. Select “use letter size instead of A4 by default” and “allow execution of restricted list of programs via write18”. Deselect all other options.

Now start the installation. When it completes, a message will be printed asking you to set up a symbolic link to the binaries. Ignore that message.

We now have BasicTeX and we want to move it to the “root” directory and then prepare that location to be used by our install package. This is done by running one script:

```
sudo sh buildPackage.sh
```

At this point, the root folder is complete and it is time to make the install package. That will be explained in a future section.

But first, let us discuss revising the script buildPackage.sh for a future year. This is easy. The only needed change is the name of the distribution, so use Find/Replace to change “2024basic” to “2025basic” or whatever in the entire document.

WARNING: I have always run “sh buildPackage.sh” using sudo. But in 2023 I discovered that this step with the GUI-Applications package causes the package to be rejected because of problems with TeX Live Utility. These problems disappear if the “sudo” is omitted. So temporarily, we use sudo on all packages except the GUI-Applications package. No other shell script described below requires sudo.

14. IMPORTANT NOTE ON BUILDPACKAGE SCRIPT

The TeX binary directory for Macintosh contains a symbolic link

```
man --> ../../texmf-dist/doc/man
```

That is because man packages on the Mac use a heuristic to find some such pages: look for a man entry in the directory containing the actual binary.

In 2019 and 2020, the configuration software which created BasicTeX and MacTeX added a third entry to texmf-dist/doc/man. This third entry was a symbolic link ”man” which again pointed to ../../texmf-dist/doc/man. There is no such location.

This spurious entry was probably created by trying to add a new ”man link” to the binary directory, but in such a way that the existing link was followed and a new link was added to its end. Therefore, after TeX is installed and before root is created, check to see if there is an existing man link. If there is, check to make sure buildPackage does not add another. Otherwise add a step in buildPackage to create that link.

In January, 2023, and in later years, the TeX Live install script adds this man link automatically and thus none of the scripts for BasicTeX-2023 adds or modifies it. But in the future, check this point.

15. INSTALL PACKAGES PART 2

Now we need to create the Basic TeX install package using the “root” directory we have just prepared. It turns out that the techniques we use for BasicTeX work the same way for all other packages. So this section explains the final steps once and for all for everything.

In the BasicTeX-2024 folder, find the subfolder CreateInstallPackage. Creating an install package from root is very simple; just change to the CreateInstallPackage folder and run three scripts as follows:

```
sh pkgbuildscript.sh
sh productbuildscript.sh
sh signPackage.sh
```

The first line creates BasicTeX-2024-Start.pkg, the second creates BasicTeX-2024-Temp.pkg, and the final line creates BasicTeX-2024.pkg. This third item will be the release package after it is notarized.

These scripts must be edited for each new year, but the editing is mainly changing 2023 to 2024 using a search and replace.

Examine the three scripts. The final script is completely straightforward. The first script is also mostly straightforward. It locates the root folder and the Scripts folder, and identifies the package with a reverse-url, in this case org.tug.mactex.basictex2024. This url must change each year and serves to keep various yearly updates separate.

The more complicated script is productbuildscript. It builds an unsigned copy of the final install package, locates the startup text in MyResources, and uses requirement.xml and distribution.xml to “design the structure of the package.” So we need to talk about these two xml files.

The file requirement.xml lists the lowest system on which the package can be installed. In 2024, that is Mojave and the file lists 10.14

The file distribution.xml gives the detailed structure of the distribution and will become more complicated when we deal with MacTeX, which is built from several subpackages. Because distribution.xml is complicated, there is a script named create-distribution.sh which can create a skeleton from the information in the Start package. This need only be run when packages are first created; later it is easiest just to edit distribution.xml. Read this file. The purpose of most lines will be clear, and other lines can just be copied and retained without full understanding (!!).

The CreateInstallPackage folder contains two important subfolders: MyResources and scripts. The first folder contains pages of text which the installer displays at the start of an install process. The second folder contains the postinstall script which runs at the end, and possibly auxiliary files used by this script. So both folders need editing at the start of each new year. In both cases, this is mainly or entirely a matter of changing dates using search and replace.

The files in MyResources are very simple. But the file postinstall in scripts is complicated because it installs the TeX Distribution Data Structure created by Jerome Laurens and Gerben Wierda. We will not discuss that structure here; read postinstall if you need to know the details.

16. NOTARIZING

The final step sends the install package to Apple. It is not touched by human hands there, but it is searched for viruses and checked to make sure all included binaries are signed and have adopted a hardened runtime. If the package passes these tests, then an official “notarization” is “stapled” to the package, which can then be released.

Notice that install packages are flat files which have already been compressed, so no compression is needed. The file can be placed directly on a server. Notarization is straightforward. Change to the directory NotarizePackage and issue the following commands:

```
sh notarytool.sh
sh stapleresult.sh
```

The file being sent is the signed install package in CreateInstallPackage, BasicTeX-2024.pkg. If this file passes notarization, the staple command adds notarization to it. So in the end, BasicTeX-2024.pkg in CreateInstallPackage will be the package released to the public.

When notarytool runs, it first checks that the package has a reasonable form. Then it prints a submission-id. This number should immediately be copied and pasted to the file submission-id.tex, because it will be needed to retrieve error information if notarization fails. Meanwhile, notarytool will be uploading the install package to Apple and printing a progress report. Uploading can take a long time for the full MacTeX package. When it is complete, notarytool will print a “processing message”, and finally a short message reporting that notarization was successful, or that there were errors.

If notarization was successful, immediately issue the “stapleresult” command and the entire process is complete.

If notarization fails, open the log.sh file and notice that the script reads

```
xcrun notarytool log "132de0cd-4584-4e0e-ab39-04f5402eebfe" --apple-id ...
```

The item in quotations should be the submission-id which notarytool printed when it started. Replace the existing string with this submission-id. Then issue the command

```
sh log.sh
```

Apple will return a long and very detailed log of notarization, clearly explaining any errors. Fix them and try again.

The good news is that notarization usually works. It only fails at the start of a "season" when errors are made revising the build process for a new year.

17. MORE ABOUT NOTARIZATION

A few years ago I wrote a short TugBoat article about notarization. However, in 2021 Apple revised the notarization tools, so the article is already out of date. Notarization in MacTeX for 2023 and 2024 is done using the new tools. These tools require a developer account at Apple, which costs \$99 a year. Another developer doing notarization will need to modify the scripts notarytool and log, because they currently refer to my account.

A glance at the scripts notarytool and log shows that each begins with the code

```
xcrun notarytool ...
```

Thus a command "xcrun" is going to run another command "notarytool" which communicates with the Apple Developer Site. For security reasons, communication with this developer site requires 2-factor authentication. However, 2-factor authentication is not convenient when running shell scripts, so Apple invented an alternative security method. A developer wishing to run a tool (in this case notarytool) which communicates with the Developer Site can log onto the site and request a password designed for that particular developer and that particular tool. Then the script can pass that password to Apple when the tool is used.

Consequently a new developer using the tool needs to log into <https://appleid.apple.com>, signing in with their developer-id and password. This will require 2-factor authentication. Then a page appears clearly listing several tasks, and one is labeled App-Specific Passwords. Select this item and a list of applications which already have a password will be shown. Add "notarytool". Apple will print a password. Immediately copy it down!. Then log out.

Now look at the text of the shell commands notarytool and log. Each contains straightforward parameters and flags, together with three crucial items:

```
--apple-id "...."
--password "... "
--team-id "... "
```

Change these to your own developer values. Recall that your apple-id is just an email address; if you log onto the developer site, the site will list your team-id.

This ends the description of BasicTeX-2024, and thus the description of most install packages in MacTeX. In the remaining documentation, we describe unusual features of the remaining folders and packages, and these descriptions will be easy reading.

Part C: Other Packages

18. MINIMALTEX-2024

MinimalTeX was requested by Vafa Khalighi. It is based on the "minimal" TeX Live scheme. I make it available on my web site, but do not upload it to TUG or CTAN.

MinimalTeX is installed in `/usr/local/texlive/2024minimal`. It contains the TeX Live "minimal" scheme.

19. TEXDIST-2024

Some users install TeX Live using the Unix install script. This includes users installing from the DVD, and users installing Mojca Miklavc's x86_64-darwinlegacy binaries for older versions of macOS. These users do not get the TeX Dist structure and cannot configure their machines to look in `/Library/TeX/texbin` for binaries.

For these users we constructed a very small install package which adds the missing TeX Dist structure. This package is called TeXDist-2024. It is available from the MacTeX web page. Construction is easy, and follows the usual pattern established by BasicTeX-2024.

20. GUI-APPLICATIONS

This package installs BibDesk.app, LaTeXiT.app, TeX Live Utility.app, and TeXShop.app in `/Applications/TeX`. In the past, it also installed a folder containing various pieces of documentation. In 2023 and 2024 this folder will be replaced by a single file `READ ME FIRST.pdf`, for users who install TeX but have no idea how to use it.

Just before we made MacTeX-2023, it was decided to add Martin Ruckert's hintview to the package so users could view HINT files. The package also contains a very short one page description of hintview.

Although this package works like previous packages, there is one tricky piece. In reality, the five applications are folders containing a large number of parts. So there is an extra step if the contents of the package change from one year to the next. In the folder `CreateInstallPackage`, there is a file named `dist.plist` which the software uses automatically to construct the package. This file must be created, and that is done automatically using the contents of `root`, by running

```
sh createdistplist.sh
```

However, the resulting `dist.plist` has one flaw. It marks each application as relocatable. In practice that means that the installer is allowed to search the user's entire machine for an old copy of each app, and then update that app. We certainly do not want that. So after creating `dist.plist`, it must be edited by hand. Search the document for keys named "BundleIsRelocatable". There should be five such keys. Each will be followed with a value `<true>`. Change each of the five values to `<false>`.

In 2023, there were problems notarizing the package due to TeX Live Utility. The author of TeX Live Utility, Adam Maxwell, has a developer account, so the program is notarized. But when we added his notarized copy to the full GUI-Applications, Apple would no longer notarize the entire package.

With great effort, particularly from Bruno Voisin, we managed to discover a work around. This requires four small changes to the usual procedure of creating GUI-Applications.

- Instead of updating TLU with Sparkle to get the latest version, update by going directly to Adam Maxwell’s site. This step is probably not necessary, but I’m too busy to check.
- In the script **buildPackage.sh**, comment out the following lines:

```
# Get TeX Live Utility
echo "Getting TeX Live Utility"
ditto ${WORK}/apps/"TeX Live Utility.app" ${WORK}/root/Applications/TeX/"TeX Live
```

- Run this script without using “sudo”.
- The result will be a **root** folder which does not contain TeX Live Utility. Now drag and drop TeX Live Utility into this folder to the spot root/Applications/TeX.

We suspect that the command line program **ditto** breaks the notarization information in TeX Live Utility, and these modifications avoid that program.

Now proceed as usual.

In 2024, we obtained TLU from Adam’s site, and manually added it to the root folder in the appropriate spot. The GUI package was immediately notarized by Apple. Thus this problem should be fixed.

21. PACKAGES WITH MULTIPLE PIECES

It is possible to create install packages with multiple pieces. The Ghostscript package has three pieces: one installs Ghostscript, one installs the library libgs, and one installs the program mutool. The last two pieces are only used by dvisvgm. The MacTeX-2024 package has five pieces: one installs GUI-Applications, one installs Ghostscript, one installs the library libgs, one installs mutool, and one installs the full TeX Live 2024 distribution.

When a package has multiple pieces, the install dialog presents an “Options” button which allows users to select which pieces to install. A user might already have up to date GUI applications, or might have a Ghostscript obtained from macports. Such a users could decide to install only the full TeX Live 2024 distribution.

If a package has multiple pieces, the developers can select the pieces which are installed by default, and the pieces which are only installed using the “Options” method.

A package with multiple pieces is constructed by first constructing install packages for each of the pieces. Then the pieces are merged together to form a single final product. This merging process is very simple. To understand how it works, look at the MacTeX-2024 subfolder. Notice that there is no `buildPackage.sh` and no root folder inside, because the full package will install the roots from its various pieces. But there is still a `CreateInstallPackage` subfolder.

This folder contains the “Start” folders for the various pieces:

```
GUI-Applications-Start
Ghostscript-10.02.1-Start
Ghostscript-10.02.1-libgs-Start
Ghostscript-10.02.1-mutool-Start
TeXLive-2024-Start
```

Each of these pieces was constructed by running `pkgbuildscript.sh`, but the additional steps

```
productbuildscript.sh
signPackage.sh
notarytool.sh
```

are irrelevant. These three scripts will be run on the final package, but not on its pieces.

In particular, TeXLive-2024 is an enormous package because it contains the full TeX-Live distribution. So making a second copy with `productbuildscript.sh` takes time and disk space and is completely unnecessary. And notarizing would involve a long, long upload to Apple, and again be completely unnecessary.

The `CreateInstallPackage` folder contains a `MyResources` subfolder because the installer will display `Welcome` and `ReadMe` panels for the entire package, not for the individual pieces. But it does not contain a `scripts` folder because the `scripts` folders of the individual subpanels are used instead. It does not contain a `pkgbuildscript.sh` because the various parts have already been made, but it contains a `productbuildscript.sh` to put together the full install package. Notarization works as before.

With this background, we finish the construction of TeXLive-2024 and of the full MacTeX-2024. We are leaving Ghostscript until the end because it is made on an entirely different schedule determined by the Ghostscript release schedule, rather than the TeX Live release schedule.

22. TEXLIVE-2024

To prepare for this package

```
cd /usr/local
sudo mv bin bin-temp
sudo mv lib lib-temp
sudo mv share share-temp
sudo mv texlive texlive-temp
```

Now install TeX Live using the TeX Live Unix install script. Call the script in the following fancy manner. Remove the `symbol` and type the last two lines as one line.

```
TEXLIVE_INSTALL_NO_CONTEXT_CACHE=1
export TEXLIVE_INSTALL_NO_CONTEXT_CACHE
cd
cd texlive2024pretest
sudo env TEXLIVE_INSTALL_NO_CONTEXT_CACHE=$TEXLIVE_INSTALL_NO_CONTEXT_CACHE \
./install-tl
```

All default options will be correct for the Macintosh except the page size option. Select “letter size”.

When the install package is created, a script *fix-for-notarization.sh* is run (see below). This script makes a small number of modifications to the package so notarization will succeed. One modification removes the Java file Arara, because it contains an unsigned precompiled universal-darwin library. In the final version of MacTeX, this file should be removed by tlmgr so the database knows it is missing and offers to restore it during updates. Such updates succeed because updates do not require notarized binaries. So if you are making a final version, switch the main repository to

```
http://ftp.math.utah.edu/pub/tlpretest
```

and issue the following command in Terminal:

```
sudo tlmgr remove --force arara
```

Note that adding the flag “-dry-run” to this call will show you what tlmgr wants to do without actually removing anything.

After users install MacTeX-2024 and run TeX Live Utility to list all packages, Arara will be listed as “forcibly removed”. TeX Live Utility can reinstall it in the standard way.

When installation is complete,

```
sudo sh buildPackage.sh
```

That step creates a new root file. Then go to /usr/local and restore the folders there.

Incidentally, the buildPackage.sh script does not create a link to man pages, since I believe TeX Live now does that automatically. It is useful to check this point.

By the time binaries reach the TeX Live pretest, they have been signed and hardened. However, a small number of files in TeX Live confuse the Apple Notarization Machinery. The shell script fix-for-notarization.sh fixes these issues. The script is short and fully commented; read it for a detailed explanation of the issues and fixes. Then issue the command

```
sudo sh fix-for-notarization.sh
```

Now create TeXLive-2024-Start.pkg as usual by calling

```
sh pkgbuildscript.sh
```

It is not necessary to call any other scripts because we only use TeXLive-2024-Start to make MacTeX-2024.

23. MACTEX-2024

Add to the CreateInstallPackage folder the following packages from other folders:

```
GUI-Applications-Start
Ghostscript-10.02.1-Start
Ghostscript-10.02.1-libgs-Start
Ghostscript-10.02.1-mutool-Start
TeXLive-2024-Start
```

Then

```
sh productbuildscript.sh
sh signpackage.sh
sh notarytool.sh
sh stapleresult.sh
```

24. COMPILING GHOSTSCRIPT

Ghostscript has become a complicated package. A small number of features of Ghostscript require X11, which is not part of macOS. But users can install the open source XQuartz to obtain X11. We want to support users who have installed XQuartz, and also users who have not. So we compile Ghostscript twice on Intel, and twice on Arm. One version supports X11 and the other does not. Eventually lipo is used to merge the Intel and Arm versions,

creating two universal binaries: `gs-X11` and `gs-noX11`. Both are installed in `/usr/local/bin` by the Ghostscript install package.

At install time, the Ghostscript install package tests the user's machine to see if X11 is installed. If it is, a symbolic link named `gs` is created pointing to `gs-X11`. Otherwise this link points to `gs-noX11`.

One program in TeX Live, `dvisvgm`, requires access to the ghostscript dynamic library `gslib`. Recently Ghostscript was rewritten, and now that library no longer provides all facilities needed by `dvisvgm`, so yet another program named `mutool` is required as well.

The `gslib` and `mutools` packages are produced by entirely different teams with different release schedules. Consequently we make separate install packages for each of them.

In short, we make three independent install packages, for `ghostscript`, `gslib`, and `mutool`. Then we create a fourth package, `ghostscript-extras`, which combines `gslib` and `mutool` in a single install module. This module is only needed by `dvisvgm`.

The creation of `ghostscript-extras` by combining the `gslib` and `mutool` packages is done in exactly the same way that MacTeX is created by combining several packages together, so there is no need to document that package here. We will provide documentation about creating `ghostscript`, `gslib`, and `mutool`.

The construction tree for MacTeX contains a folder named `Ghostscript`. This folder contains all subfolders and material needed to construct the two install packages just described.

Just as MacTeX-2024 was being completed, a new test release of Ghostscript 10.03 was provided. This version will not be in MacTeX-2024 because the test release is not yet the official release. But we provide separate install packages for it. When we created those packages, we slightly revised the build methods for Ghostscript, and so this documentation will describe the revised build methods. We began by creating a folder named `Ghostscript-New`, which was mainly just a copy of `Ghostscript` but for release 10.03. It is this `Ghostscript-New` folder which we document below. For simplicity, we name the folder "Ghostscript" in these notes because it will become the `Ghostscript` folder for all future versions of MacTeX.

25. GHOSTSCRIPT-BINARIES

The first step in creating Ghostscript is to compile the binaries. Inside `Ghostscript` is a folder named `GhostscriptTransfer`. This folder will be copied to a small transfer hard disk, and moved via that disk to the Arm and Intel machine used to actually compile binaries. So the folder contains all information needed to compile the various pieces of Ghostscript.

The folder contains two empty folders named Binary-Arm and Binary-Intel. These folders will ultimately contain the binaries created on the Arm and on the Intel machine. Four arm binaries are created, named

```
gs-X11-arm
gs-noX11-arm
libgs.dylib.10.03-arm
mutool-arm
```

The corresponding intel binaries have the same names, but end with the word “intel”.

The folder also contains this document, so it can be referred to during compiling. It contains two tar files, one with the Ghostscript source and one with the muTool source.

The Ghostscript sources can be found at <https://ghostscript.com/releases/gsdnld.html>. Select the Ghostscript AGPL Release. The site <https://github.com/ArtifexSoftware/ghostpdl-downloads/releases> contains these files as well, and also has Release Candidate code for pre-releases.

The muTools code can be found at <https://mupdf.com/releases/index.html>. Look for a file named approximately “mupdf-1.23.11-source.tar.gz”. This source builds both muPdf and muTools, but we only take the second binary.

Finally the GhostscriptTransfer folder has an option Version folder, which can be used to record the latest versions and release dates for the software obtained. This information is useful when writing the ReadMe file for the final install package.

26. BUILDING THE BINARIES

The binaries are built in exactly the same ways on Arm and Intel machines. We’ll discuss the Arm case here. Copy the entire GhostscriptTransfer folder from the transfer disk to the Arm machine. Open the copy of this document to follow these instructions.

Using Terminal, issue the following commands:

```
cd /usr/local
sudo mv bin bin-temp
sudo mv lib lib-temp
sudo mv include include-temp
sudo mv share share-temp
```

Uncompress the source files for ghostscript, and for mupdf. Make sure that X11 is available on both machines; it need not be running. Compile ghostscript twice, once with X11 available and once with it disabled. To check that X11 is active,

```
cd /opt
ls
```

The result should be “X11”.

With X11 active, build ghostscript using the commands

```
make clean
./configure --disable-compile-inits
make
```

Find the resulting binary gs in the ghostscript-10.03 directory and rename it gs-X11-arm. Copy this to the GhostscriptTransfer/Binary-Arm folder.

Next issue the command

```
make so
```

When this completes, go to the sobin directory in ghostscript-10.03 and find a library file and two symbolic links. Rename the library file libgs.dylib.10.03-arm and copy it to the GhostscriptTransfer/Binary-Arm folder.

Next disable X11

```
cd /opt
sudo mv X11 X11-temp
```

Compile ghostscript again using the above commands and rename the resulting gs file to]gs-noX11-arm and copy it to the GhostscriptTransfer/Binary-Arm folder. Then activate X11 by

```
cd /opt
sudo mv X11-temp X11
```

Finally, compile muTool by changing to its source directory and issuing the command

```
sudo make HAVE_GLUT=no HAVE_X11=no install
```

When this is complete, find the muTool binary file in \usr/local/bin, rename it to muTool-arm and copy it to the GhostscriptTransfer/Binary-Arm folder.

Finally fix /usr/local by

```
sudo rm -R bin
sudo rm -R lib
sudo rm -R share
sudo rm -R include
```

followed by

```
sudo mv bin-temp bin
sudo mv lib-temp lib
sudo mv share-temp share
sudo mv include-temp include
```

When the same steps have been done on the Intel machine, move the transfer disk back to the computer used for final building, which this year is Sonoma. Copy the contents of GhostscriptTransfer/Binary-Arm and GhostscriptTransfer/Binary-Intel to the corresponding directories in Ghostscript/GhostscriptTransfer.

27. LIPO FOR GHOSTSCRIPT AND MUTOOL BINARIES

Now we are back on Sonoma, finishing the construction of the Ghostscript binaries. The Ghostscript folder contains a file named liposcript-gs-mu.sh. Run this file by changing to the Ghostscript directory and then

```
sh liposcript-gs-mu.sh
```

The script will remove the old Ghostscript/Binaries folder and rebuild it using the binaries in GhostscriptTransfer/Binary-Arm and GhostscriptTransfer/Binary-Intel. The result will be a file name Binary inside Ghostscript containing signed universal-darwin binaries

```
gs-noX11
gs-X11
libgs.dylib.10.03
mutool
```

These are the four binaries used by the various Ghostscript packages.

28. GHOSTSCRIPT-10.03

Next we build the Ghostscript install package. This requires compiling Ghostscript one more time and installing it temporarily in /usr/local. So begin by

```
cd /usr/local
sudo mv bin bin-temp
sudo mv lib lib-temp
sudo mv share share-temp
```

Next make sure that X11 is active. Then switch to the Ghostscript source code and issue the commands

```
make clean
./configure --disable-compile-inits
make
sudo make install
```

Now go to the folder Ghostscript/Ghostscript-10.03. Issue the command

```
sh buildPackage.sh
```

This command creates the root folder for the Ghostscript Install Package by copying Ghostscript from /usr/local/bin and /usr/local/share to the root folder. The gs

file in this root is removed and replaced by the universal-darwin binaries gs-X11 and gs-noX11.

After this step, clean up `/usr/local` by

```
cd /usr/local
sudo rm -R bin
sudo rm -R share

sudo mv bin-temp bin
sudo mv lib-temp lib
sudo mv share-temp share
```

Now finish the Ghostscript-10.03 install package by running `pkgbuildscript`, `productbuildscript`, `signpackage`, `notarytool` and `stapleresult` in the standard manner.

29. GHOSTSCRIPT-10.03LIBGS

Go to the folder `Ghostscript/Ghostscript-10.03libgs`. Issue the command

```
sh buildPackage.sh
```

This command creates the root folder for the Ghostscript libgs Install Package. Create the package by changing to the directory `CreateInstallPackage` and proceeding in the standard manner.

30. GHOSTSCRIPT-10.03MUTOOOL

Go to the folder `Ghostscript/Ghostscript-10.03mutool`. Issue the command

```
sh buildPackage.sh
```

This command creates the root folder for the Ghostscript mutool Install Package. Create the package by changing to the directory `CreateInstallPackage` and proceeding in the standard manner.

31. GHOSTSCRIPT-10.03-EXTRAS

Erase all previous install packages in the folder. Put copies of `Ghostscript-10.03-libgs-Start.pkg` and `Ghostscript-10.03-mutool-Start.pkg` in the folder. Then run `productbuildscript`, `signPackage`, `notarytool`, and `stapleresult` in the standard manner.

32. GHOSTSCRIPT SUMMERY

In the end we obtain two install packages, `Ghostscript-10.03.pkg` and `Ghostscript-10.03-Extras.pkg`.

33. XMACTEXDOC

This folder contains the instructions needed to create MacTeX and related packages. It has this READ-ME document, together with a skeleton of the full build file tree. The skeleton contains scripts (with passwords removed), but omits large files created during the build process.

After revising this folder,

```
sh compress.sh
```

The result will be a file named `mactexdoc-2024.tar.xz`. Send this file to Karl Berry, who will install it in the Master/Source directory.

34. MISCELLANEOUS

Tex Live Utility updates of the pretest are available during testing, but a pretest repository must be used. I use

```
http://ftp.math.utah.edu/pub/tlpretest
```

I recently discovered how to determine if a binary file has been signed by a developer, and if so who and when. Type the following in Terminal:

```
/usr/bin/codesign -d -vvvv
```

and add a space after the `vvvv`. Then drag and drop the binary file into Terminal. This last step will add a complete path to the binary to the end of the Terminal command; this step can also be done directly if desired. When the command is executed, a large amount of information will be provided, including the name and program group of the developer and the date of the signature.