Configuring OS X for command-line programs

These instructions are written for users of the CDP Multi-Channel Toolkit who have not installed the full CDP system, and who have not used command-line programs before. CDP users may also find it relevant, if they want to use the system from the command line.

Underneath the slick graphics, OS X is a species of Unix. Some tasks can only be performed using a text console, and by typing in instructions. In OS X, this text console is called "Terminal", which can be found in **/Applications/Utilities**. It is recommended that this be made accessible from the Dock: simply drag "Terminal" to the Dock. OS X also provides a text editor, called "TextEdit", which can be found in **/Applications**. Drag this to the dock in the same way. Note that by default it creates new documents in RTF format; you will need to change this to Plain Text format via TextEdit's Preferences page.

Toolkit users are faced with three basic tasks:

- configuring Terminal to find the various command-line programs in the Toolkit
- Navigating through directories containing soundfiles
- running Toolkit programs to process or play soundfiles

Other tasks might include:

- copying soundfiles
- moving soundfiles between directories
- deleting soundfiles
- renaming soundfiles

These tasks can all be performed from the command-line. In more advanced usage, wildcard characters can be used to perform multiple changes with one command. For more complex batch processing shell scripts (offering powerful programming facilities) can be used, taking full advantage of the underlying unix-like nature of OS X.

Clicking on the Terminal icon in the Dock will launch Terminal, which will display a window to receive text commands and print any text output from them.

The first thing to note is the title: this will indicate one of two forms of **command "shell"**:

bash
tcsh

The difference between these is only of concern to advanced users experienced in using Unix tools. However, the syntax for some commands is different, and the private initialization files are different, so it is important to know which shell you are using. On all modern Mac systems, Bash is the default shell.

Open a Terminal session: you will see a command prompt comprising the computer name and your user name, for example:

bash shell:
`cdpmac:~ fred$`

tcsh shell:
`[cdpmac:~] fred$`

At this stage, you are "in" your **home directory)**. The following instructions assume that you stay in this directory.

A standard Unix shorthand for "my home directory" is the tilde character: **~** which as shown above forms part of the initial shell prompt.

The CDP system, including the multi-Channel Toolkit, comprises a large number of programs that are **not** full Applications that one can double-click on from the Finder, **but rather are command-line tools**. Front-end applications such as *Sound Loom* run them behind the scenes; but they are actually designed to be run directly from the command-line, i.e. from within a Terminal session. To access them, Terminal needs to know where they are – what their "**path**" is – i.e. what directory they are in. This in turn means that we may want to create a directory in which to put such programs. The natural procedure of course is to use the Finder to do this – e.g. to create a folder called "bin" in your home directory. It is equally possible to do this directly from the Terminal.

All commands comprise a (usually) short command name followed by any further arguments (separated by spaces), and completed by pressing the Enter key, which initiates the command. In the examples below, all text typed by the user is shown in a monospaced font.

Some basic unix commands:

List directory:
`ls`

List Directory in a "long" format that displays the size and modification date of each file (or directory), and the permissions setting for each:
`ls -l`

Make Directory "dirname":
`mkdir dirname`

Change into Directory "dirname":
`cd dirname`

Change to your home Directory:
`cd ~`

Read environment variables:
`env`

Note especially the **PATH** string reported by this command

To set up Terminal to run the Toolkit programs we must place them in a convenient directory, and then add the full path name of that directory to the user's PATH as shown by the **env** command.

A conventional name for a directory to contain executable programs (also termed "binary" files in unix-speak) is simply **bin**. You will find that some system directories on the system already have this name. These directories are not visible in the OS X finder (nor from any GUI application), but are accessible from a Terminal session. For example: type the command **ls** followed by a space and a forward slash:

```
ls /
```

This asks the shell to list all the directories from the root of the whole system: these directories will include all the standard system directories familiar to any unix user. Similarly, to see the contents of the directory called `/usr` type:

```
ls /usr
```

A typical output would be:

```
X11R6           include         libexec         sbin
standalone
bin             lib             local           share
```

In principle, the Toolkit programs could be placed in any of the available system bin folders, with
/usr/bin or **/usr/local/bin** the most likely locations. Writing to any of the system folders requires the use of the administrator password (and shell commands must be prefixed with the special command **sudo**). This is not recommended for users inexperienced in the command-line environment.

Instead, we will see how to create a new **bin** folder inside the home directory, and how to add its path to the list searched by the shell. Each shell checks a special hidden initialisation file when it starts up - the name of this file is particular to each of the available shells. These files are not present by default in a new OSX system – we will need to create them ourselves.

Open a new Terminal session, and note whether you are using the **bash** or **tcsh** shells; currently **bash** seems to be the default. At the command prompt, type the following command to create the new directory:

```
mkdir bin
```

It is easiest at this stage to use the finder to copy the unpacked Toolkit programs to the new directory.

To create the bash initialisation file:

1. If you do not have a file called `.bash_profile`.

this is a file read by the Bash shell when a new terminal session is started. It is not visible in Finder because the leading dot character marks it as a "hidden" file. So how do we find out? Make sure you are in your home directory, as described above (symbolised by the ~ character).
Then type the command:

```
ls -a
```

This prints a directory listing including the "dot" files - there may be several (the -a flag asks `ls` to display *all* files). The following instructions assume that `.bash_profile` is not present, so we have to create one.

Open the OS X application TextEdit to create a new text file. Write the following text exactly as shown:

```
PATH=$PATH:~/bin
```

Use File->Save As to save the file with the name `.bash_profile` in your home directory. In the Save As dialog, uncheck "If no extension is provided, use .txt". When you click Save, a further dialog will appear with a warning about using the dot name - click "use .".

To confirm that the file has been correctly named and saved, go back to your Terminal window and type the list command again:

```
ls -a
```

2. If you do have `.bash_profile`.

This will have been created by some other program or library that you installed. It may or may not already have a PATH definition. You do not want to overwrite this file, or you will lose important configuration setting for whatever software created the file. This now includes the CDP system!

You can open a file from Terminal in the default text editor (e.g. TextEdit.app) simply by typing:

```
open .bash_profile
```

Or, if you are comfortable with the *emacs* editor, open the file in that directly:

```
emacs .bash_profile
```

All we have to do is add our path to any existing one. In fact, exactly the same line can be used as shown above:

```
PATH=$PATH:~/bin
```

as this appends our path to the existing one; or you can add the new path directly to the end of the existing PATH string, remembering to use the separator character, here a colon. Note that the system defines a default PATH early on in the boot process, inherited by each user. It is always essential to use an "update" PATH command as shown above; otherwise we will lose the paths to all the commands upon which Bash itself depends!

To create the tcsh initialisation file:

Create a new file in TextEdit, entering the text below, exactly:

```
setenv PATH ${PATH}:${HOME}/bin
```

Again, you will need to confirm that the file `.tcshrc` does not already exist - use the same procedure as described above. Assuming it does not, save this file with the hidden name **`.tcshrc`**. Otherwise, edit the file to add the new path at the end of any existing PATH statement.

The different text in the two files reflects the differences of language between the two shells. However, the action is the same - to read the existing PATH string (as shown by the `env` command), and append the new path.

NB: As their name suggests, these initialisation files are only read when a Terminal session is launched, so **to activate them** you will need to close all Terminal Sessions (you can have more than one running at the same time!).

It is also possible to run one shell from another, simply by typing the name of the required shell. If you do this, you return to the previous shell by typing **`exit`**.

Using the command line to run Toolkit and programs.

To test the installation it will be useful first to change to a folder containing some soundfiles. Assuming you have a folder "**sounds**" in your home directory, use the **Change Directory** command "**cd**" to change to it:

```
cd sounds
```

The directory sounds is now your "current directory". all commands operate relative to the current directory.

A trick with cd
You can obtain a directory or file path by dragging it from the Finder into the open Terminal window - the path will then appear at the prompt line. The trick is to type `cd` and a space

first, then do the drag; the command is then ready to be invoked. This is especially useful for paths to some deeply nested directory - saves typing and avoids those typing mistokes!

Now run some programs, firstly by themselves:

```
sfprops
```

This will list the properties of a soundfile. Used without a filename it will print a usage message. Read this, and then run sfprops again, giving it the name of a soundfile:

```
sfprops testfile.aiff
```

Other programs take one or more command-line arguments, separated by spaces. To play the soundfile *testfile.aiff*, use the Toolkit program **`paplay`**:

```
paplay testfile.aiff
```

If the file is mono, you could create a surround version panned around the listener using **`abfpan`** `or` **`abfpan2`**. Type the name of the program by itself to obtain the usage message. To rotate the file two times around the listener, type the following:

```
abfpan testfile.aiff  testpan.aiff 0 2
```

To play such a multi-channel file using **`paplay`**, it will of course be necessary to have an audio device that can render to the required number of channels.

Alternatively, you can try out the AMB file format using the `-b` flag, and giving the output file the *.amb* extension:

```
abfpan -b testfile.aiff  testpan.amb 0 2
```

Note that the new program **abfpan2** only outputs a (second-order) AMB file, it does not decode. Use the new program **fmdcode** for this (or **paplay**).

Some standard command-line operations.

To copy files from the current directory to a sub-directory called "aiffsamples", use the command **`cp`:**

```
cp *.aiff aiffsamples
```

The '**wildcard**' character ***** means "anything", so this says copy **all** files with names ending with **.aiff** to the folder **aiffsamples**. If you have a set of piano samples will a common root name (e.g. *pianoA1.aiff*, *pianoA#1.aiff*, *pianoB1.aiff* etc.) , you can use the wildcard mechanism to copy just those files to a new subdirectory **`piano`**:

```
mkdir piano
cp piano*.aiff  piano
cd piano
paplay pianoA1.aiff
```

You may be impressed how speedily these operations are performed, compared to the manoeuvres required to do it from the finder!

The move command is similarly simple. This moves all files whose names start with "piano" to the new directory:

```
mv piano*.aiff piano
```

The `mv` command is also used to rename files:

```
mv verylongfilename.aiff  vname.aiff
```

Or in combination, to move the file to the new folder:

```
mv verylongfilename.aiff  piano/vname.aiff
```

Note however that these commands will not work quite so conveniently with file or directory names containing spaces. Such names have to be enclosed in quotation marks:

```
mkdir "Tenor Sax"
mv "sax A 1.aiff" "Tenor Sax"/saxA1.aiff
```

Such filenames are really somewhat inimical to fluent command-line work, simply because spaces are universally interpreted as string separators. Users likely to make extensive use of the command-line will find it expedient to rename such files to names replacing spaces with, say, the underscore character. An adept unix programmer would create a shell script (a text program run by the shell itself) to perform this task for a large number of files; at this stage it will be safer to perform the task by hand in the Finder. Worst of all are names containing multiple spaces; it is all but impossible to work with these from the command-line (at least, without investigating each filename individually from the finder first!).

To delete a file (or files), use the **`rm`** command:

```
rm testfile.aiff
```

Wild-cards can be used:

```
rm piano*.aiff
```

But take care! The command:

```
rm *
```

will remove *every file* in the current directory, with no possibility of recovery, and with no helpful warning message!

Finally, note that all such commands have **online help** available in the form of the "man" command.
Typing:

```
man mv
```

will print a stream of text information to the console (press the spacebar to read the next page, enter to move down a single line, and type q to return to the shell prompt).

To see just how comprehensive this documentation can be, try typing:

```
man bash
```

This will print the full documentation on the bash shell, including the syntax and keywords used for advanced shell programming.

Needless to say, this document only scratches the surface of what is possible from the command line. OS X users are encouraged to research further, perhaps by buying one of the many books available on how to use a unix system (some are specific to OS X), how to write shell scripts, and how to use other scripting languages such as perl or Python, both supplied as standard with OS X.

Richard Dobson, July 2006, updated November 2010