2. A quick window manager and Unix tutorial

(This section last updated August 2012)

This section provides

• some "getting started" pointers on the graphical *windowing environments* you will be using, primarily on the ECMC Linux system in the room 52 studio, but also with some references and comparisons to comparable Mac OSX and Windows procedures; and

• an introduction to some basic Unix commands and utility programs that can be run from a shell window on ECMC Linux and Macintosh music production systems

As noted in section 1, the core of a computer operating system, which implements the most basic system protocols and operations, is called the *kernel*. The open source Linux kernel is developed and maintained by an international group of thousands of volunteers under the direction of Linus Torvalds, who wrote much of the first Linux kernel, modeled after the functionality of Unix operating systems, during the early 1990s. The Linux kernel is updated several times each year, and includes both a *stable* branch, designed for most users (like us), and a *developmental* branch, used by Linux developers to test new features before they are incorporated into stable versions of the kernel.

Technically, the term "*Linux*" refers only to an operating system *kernel* developed by the team of Linux Project programmers under Torvolds' direction. The *GNU* (pronounced "*ga-NOO*") *Project* is a non-profit organization that supports the development of a broad range of open source applications, without which the Linux kernel would be useless. Many users of GNU software on Linux kernels prefer to call their operating system *GNU Linux* or *GNU/Linux*. In common usage, however, the term *Linux* often is used loosely to refer to all GNU operating system components as well as the Linux kernel. Usage of GNU Linux software is rapidly replacing more expensive, closed source proprietary versions of Unix.[1]

Beginning with *Mac OS X*, the Macintosh operating system has been based on a proprietary *mach* kernel, which was originally used on computers manufactured by the NeXT corporation during the late 1980s and 1990s, and which also is based on a Unix model. Thus, "under the hood," Macintosh and GNU Linux systems are remarkably similar. Versions of Microsoft's *Windows* operating system are based upon a core *Windows* kernel, with some of the same basic functionality as Unix and Unix-like kernels, but a significantly different design philosophy and implementation.

In addition to the low level kernel, a complete, usable *distribution* of an operating system includes a much larger base of software that is built on top of the kernel, including such components as

• programs that enable users to access and manipulate files
A majority of the basic GNU/Linux utility programs are commands that can be issued from a shell window. Most of these commands (many of which are available on Mac OSX as well) are derived from AT & T *System V* versions of Unix and/or from *BSD* (Berkeley Standard Distribution) versions of Unix that were written in the 1970s, 1980s and 1990s.

• *applications* : generally larger, multi-purpose programs or suites of related programs that include a graphical front end for submitting and displaying data

• *device drivers* that enable the OS to control particular peripheral devices (printers, sound cards, graphics boards and so on)

• compilers for languages such as *C*

• protocols for a graphical windowing environment, so that applications written for the OS employ consistent mouse-based procedures and graphical aids for opening and manipulating windows and files.

• networking software to connect to other computers

• developers tools — additional protocols for programmers who wish to write third party applications usable on the OS

[1] For more information on the history of GNU software, the Linux kernel, the *GPL* (General Public License) under which GNU software is distributed, and contrasts between open and closed source software paradigms, see *http://www.gnu.org/gnu/thegnuproject.html* or *http://web.dodds.net/˜hozer/dream.html*.

2.1. Window manager interfaces

Windows is distributed by the Microsoft Corporation in various "editions" for client and server machines used by businesses, individual users and software developers. All Windows distributions provide a consistent graphical **window manager** (or "*desktop manager*") interface through which users can open and control multiple windows on the monitor screen, launch and quit applications, find their way around the system's resources, and adjust certain system settings and preferences.

Apple's Mac *OSX* also is sold in several distributions designed for differing purposes, although generally for client machines and standalone workstations rather than for cross-platform servers. To assure consistency in how applications look and work, Mac OS X employs an underlying graphical protocol system known as *aqua*. The most fundamental application that employs these aqua protocols is the *Finder*, the *window manager* and file navigation system employed on all Macs. Other commercial distributions of Unix, sold by vendors such as IBM and Hewlett-Packard, primarily for servers, also provide their own *desktop manager* applications, but these are based instead on *X11* (also known as *X Windows*) graphical protocols originally developed for Unix systems at MIT.

With Linux, as is often the case, the situation is more complex. Linux distributions are available for sale (with technical support) or by free download (without vendor tech support) from commercial companies such as *Red Hat*, *Canonical* (*Ubunutu* and *Kubuntu*) and *Suse* (owned by Novell), as well as from non-commercial associations such as *Fedora* (a "developmental" platform supported by Red Hat), *Debian*, *Gentoo*, *Slackware* and many others. Some of these Linux distributions include a great deal of bundled software requiring many gigabytes of storage space for full implementation. Other distributions are designed instead to be lean and fast.

All Linux systems employ *X11* windowing protocols for graphic displays. However, there is no single, universal Linux *window manager* application. Rather, alternative *desktop manager* applications (different implementations of the *X11* protocols) are available from various teams of Linux developers. Some people view these options in the "look and feel" of the user interface as a strength for Linux (facilitating customization and choice) while others have viewed it as a drawback ("lack of uniformity" or "complexity"). The two most widely used of these Linux window managers — both "full-featured" variants — are:

- the *Gnome* desktop and development platform, developed primarily by a US team of programmers for use on most Unix-like systems
- *KDE*, developed primarily by a European Linux team, which has a "look" and functionality similar in some respects to that of the Microsoft *Windows* "desktop environment"; and

Two other popular alternative Linux window managers include *Xfce* and *Fluxbox*.

In addition to somewhat different windowing conventions, KDE and Gnome also provide complete (but different) sets of applications suites, including word processors, spreadsheets, CD players, and so on. However, it is possible to run most KDE applications under Gnome, and vice versa, so long libraries for both are installed on a system.

On ECMC system *madking* we have at one time or another employed Red Hat, Suse and Mandrake distributions, and have used both KDE and Gnome as our default desktop environment. The differences between these "distros" and GUI environments are not really all that significant, and users can adapt quickly from one to another. As of this writing we are running a *Fedora* distribution on *madking* with the *Gnome 3* desktop. While not our favorite in all respects, the *Fedora* distribution makes it possible, in conjunction with an audio application bundle called *PlanetCCRMA At Home*, maintained by Fernando Lopez-Lezcano at Stanford University, for us to maintain and continuously update an extensive package of open source audio software in an easy, largely automated fashion, so that we are always running current stable versions of these applications. Another layer of music/audio software on *madking* consists of the ECMC *Turnkey* package — a collection of Linux shell programs and utilities written over the past 30 years by ECMC users.

Users who have previously worked on Macintosh or *Windows* systems will find many familiar concepts and techniques within the *Linux* desktop environments. Many graphical display and selection conventions have become standardized across all major computer platforms today.[2] In fact, for many general

[2] Many of these graphical conventions were originally developed at Xerox research centers in Silicon Valley

purpose computing tasks (but definitely not for all of our music-making purposes), it is possible to use Unix-based systems like *madking* without ever issuing a Unix shell command, or even knowing much about Unix, by exclusively using the point-and-click tools provided by the *window manager* environment and graphical software applications. However, this definitely is not recommended if you want to harness the full power of a Linux system.

In computer parlance, an **application** (or *"app"*) is an interactive graphical program that makes use of the graphical tools and conventions provided by the user interface of an operating system. On *Windows* and many *Macintosh* systems, practically all tasks are performed by means of *apps* provided by the operating system or obtained from commercial software vendors or freeware and shareware sources. Some applications run only on a single operating system — for example, the simple Windows word processor *Notepad* and its Mac counterpart *TextEdit*, or the Mac-based music sequencing program *Logic*.) Other applications are cross-platform, available for use on two or more operating systems. The *Firefox*, *Chrome* and *Opera* web browsers, for example, can be used on Windows, Mac or Linux systems; the *Cubase* sequencing program and *Reaktor* software synthesizer are available for both Mac and Windows systems. Many people and organizations select computer hardware and operating systems based largely on which applications they support. At the ECMC, by contrast, most of us prefer cross-platform applications that free users from dependence upon a particular operating system.

Mousing around the desktop

Most Macintosh, *Linux* and *Windows* systems employ three button mice, which perform different functions (although there is some overlap and duplication).

☞ the **left** mouse button is the most important of the buttons, and generally is used to open applications, to make selections and to move windows around on the desktop. When you are instructed to click on or to drag the mouse over an icon or to make selection an item, this should be done with the left mouse button.
Windows and Mac applications most often are opened for use by double clicking with the left mouse button on the application's **icon** (a small graphical image on the monitor that represents the application). With Linux, however, by default single clicking on an icon opens the application. If you click twice, you will open two copies of the application.[3]
☞ the **right** mouse button pops open a hidden context menu with options for changing the appearance of the window or performing certain other common tasks. systems, clicking the right mouse button in any blank space on the monitor screen (between windows) opens a menu with options for creating new documents or for arranging icons or the desktop. Right clicking on an open window provides options for manipulating this window, its contents, or user preferences.
☞ the **center** mouse "button" is a scroll wheel that can be used to control vertical scroll bars within a window.
☞ *Linux* only: the **middle** mouse button is most often used as a *paste* button, to "paste" text that has been previously copied to the "clipboard" buffer by dragging with the left mouse button to select (highlight) a portion of text. Text can be copied between any two windows, or between different locations within a single window, in this fashion, by highlighting the desired text, then by clicking at the desired insert point with the middle button. It is not necessary to perform *Ctrl-c* and *Ctrl-v* (or *Shift-Ctrl-c* and *Shift-Ctrl-v*) to copy and paste.

Recent changes in desktop environments

The growing popularity of small, portable, often hand-held computing devices, including netbooks (small, light, inexpensive laptops) and tablets (like the *iPad*), media players (such as the *iPod*) and smartphones has necessitated the development of new operating systems with simplified user interfaces. Since these devices often lack a mouse or trackpad and generally substitute a reduced virtual keyboard for a physical keyboard, interface controls such as onscreen buttons, sliders and switches are manipulated by multi-touch finger movements (tap, swipe, pinch, scroll, flick, etc.) to access programs, make selections and input data. The most common of these simplified operating systems are Apple's *iOS*, Google's Linux-based

during the early 1980s, subsequently were popularized and extended in releases of the Macintosh *Finder* OS, and then were emulated by Microsoft, Unix vendors and Linux developers.

[3] However, you can change this single click default to double clicking if you like.

Android, and *Windows Phone* and *CE*.

To most hardware and software vendors it seems inevitable that a majority of computer users increasingly will employ portable hand-held devices in place of desktop and laptop computers for routine consumer-oriented tasks such as email access, web browsing and music and video playback. As a result, software developers recently have begun attempting to provide consistent, "universal" user display and control interfaces for a wide range of consumer computing devices, from smartphones through laptop and desktop computers, and have ported features from hand-held systems to recent versions of *Windows*, *OSX* and Linux. Displays and controls have been "simplified," often designed for the display of one window at a time.

Apple has incorporated interface features from *iOS*, such as fullscreen mode, a stripped-down application lanucher (*Launchpad*) and an App Store, into recent versions of *OSX* ("Lion" and "Mountain Lion"). *Windows 8* represents a first step in Microsoft's more radical purported plan to develop a single operating system suitable for use on a wide range of systems ranging from small tablets to larger tower PCs; user interface and display conventions common in Windows since the mid 1990s are being supplemented or replaced by a touch interface called *Metro*. These changes have been controversial, especially with regard to *Windows 8*. Many "power" users have complained about "reduced functionality," "change for change sake" and "extra effort to accomplish simple tasks."

Gnome fallback mode

The most far reaching interface and display redesign to date, however, can be seen in the *Gnome 3* Linux desktop environment, released in 2011, which is bundled as the default windowing system of *Fedora* Linux distributions that we use on *madking*. Where version 2 of *Gnome* employed fairly standard windowing and display conventions, the *Gnome 3 shell* jettisoned many of these conventions in favor of a "simplified," "streamlined" interface. The default *Gnome 3* interface has no taskbar or desktop workspace, no maximize or minimize buttons on window borders, no place to put launcher or folder icons, no trash can, reduced right-click capabilities, not even easily accessible "shutdown" and "restart" tabs.[4] Instead, the minimalist standard *Gnome 3* screen contains a single *Activities* trigger. Clicking on this trigger displays a launcher, a taskbar- or dock-like panel for pinning frequently used applications, and a display of open windows spread horizontally, possibly over several contiguous workspaces.

Whether or not the *Gnome 3* design reflects the future of desktop computer interfaces or, in the disgruntled words of Linux Torvolds, "an unholy mess," remains to be seen. Several Linux distributions, including *Ubuntu*, have dropped Gnome as their default windowing system. For users (like me) who find traditional desktop conventions more intuitive and productive, however, *Gnome 3* does provide a more familiar *fallback* mode, which I recommend that you use for all of your work sessions on *madking*.

To use *fallback* mode:
When you log on to *madking*, after clicking on your USERID (login) name, but before typing in your password, click on the $\boxed{\text{Session}}$ tab, and, from the list of available windowing systems, select
Classic Gnome with Compiz.
This will implement *fallback* mode for the current and all future sessions until you change your *session* type again some time in the future.

The next few pages of this section will assume that you are using Gnome in *fallback* mode. As of this writing, *fallback* mode is quite new, and it is by no means perfect. Right-click functionality remains reduced, and the *Help* documentation is incomplete. There may soon be significant improvements, or we eventually might decide to drop Gnome entirely and employ some other windowing system on *madking*.

Accessing system resources: taskbar (dock) panels

To run an application or open a file, you first must *find* the application or file so that you can launch it. This can be done, sometimes rather laboriously, by clicking on the window manager's *starting point* (e.g. the *Start* button in *Windows* or the *Apple* or *Finder* tab on Macs), which will open an exploding series of menus or tabs, then finding your way to the folder where the *application* or file is stored, and then finally launching it.

[4] Many of the features actually are available, but only by means of special keystroke combinations. "Shutdown" and "reboot" commands, for example, can be displayed by pressing $\boxed{\text{Alt}}$ $\boxed{\text{F2}}$.

Alternatively, for quick access to applications and files that they will use frequently, users can create a permanent pointer to the application or file on the main desktop display. This pointer is called a *shortcut* on *Windows* systems and an *alias* (or *link*, or *launcher*) on Unix-based systems. Some users place many shortcut aliases directly on their desktop — convenient, but not a good idea, since these icons frequently must be redrawn by the system, tend to clutter the desktop and often will be hidden by windows for running applications. Most window managing system provide a better place for application and file launchers — a horizontal (or, less often, vertical) panel along one edge of the monitor display, on which users can "pin" application and file launchers.

☞ On *Windows* systems a horizontal *taskbar* is located along the bottom of the monitor. A *System Tray* at the far right of the taskbar contains icons for important utilities and system settings (e.g. a clock and an anti-virus application).

☞ On *Mac* systems provide a *dock*, a strip of large icons usually located along the bottom of the desktop, on which users can place frequently used applications. The *dock* can be manually hidden or displayed (Command+Option+D) or can be automatically hidden when not in use, a userful feature on smaller screens.

☞ On the Linux *Gnome fallback* desktop you probably will have two horizontal panels (taskbars), one across the top of the screen and another across the bottom, which displays tabs for currently open applications and windows. Clicking on one of the lower panel tabs makes it the active (selected) window.

The top panel is similar to a *Windows* taskbar. Clicking on the **Applications** menu in the upper left corner will open a list of applications and programs available on the system organized by category.

- You can obtain help on the desktop environment by selecting *Applications->Accessories->Help*.
- The *terminal* application for running shell commands is found in the *System Tools* group.
- *Applications->Accessories->Application Finder* enables you to search for applications.
- *Applications->Sound & Video* contains launchers for frequently used audio applications, such as *Audacity* and *QjackCtl*.

Clicking on the **Places** menu (to the left of the *Applications* menu) will display shortcuts to your home Unix folder and available disk partitions.

At the far right of the top panel, clicking on your *USERID* will open a menu with options including *Log out*, *Restart* and *Shutdown*.

The remainder of the top panel can be used for placing icon aliases for particularly important applications, directories and files. All operations on this customizable portion of the top panel are made by holding down the WIN key (the key with the *Windows* logo, usually adjacent to the left side *ALT* key) and the ALT key simultaneously while clicking with the right mouse button. To add an application launcher, directory or file to the panel, press WIN and ALT while clicking and select *Add to Panel*.

To add a *Terminal* (*shell*) launcher icon to the panel:

- Right click while holding down the WIN and ALT keys
- Select *Application Launcher*, then click Forward
- Click the > arrow next to *System Tools* to display available tools. From this list, click on *Terminal* and then on Add

To move an icon to another position on the panel, WIN + ALT + right click on the icon and select *Move*. Then, with the left mouse button, drag the icon to the desired position.

To remove an icon from the panel, WIN + ALT + right click on the icon and select *Remove from panel*.

Application launcher icons that you probably or may wish to place on your Gnome top panel include:

- a terminal (Unix shell) icon (see instructions above)
- an icon for *QjackCtl*, used to configure, launch and stop the *Jack* sound server (available in *Application Launcher->Sound & Video*)
- icons for web browsers, such as *Firefox* and *Chrome* (available in *Application Launcher->Internet*)

Icons to folders that you also may wish to place on your Gnome top panel for quick access include:

- an icon for your home directory (/home/USERID)
- an icon for your home soundfile directory (/snd/USERID)

• an icon for for the sflib and sflib96 directories (/sflib and /sflib96)

To add these directory shortcuts to the panel:

• $\boxed{\text{WIN}}$ + $\boxed{\text{ALT}}$ + right click on the panel and select *Add to Panel*
• Select *Custom Application Launcher*, then click on $\boxed{\text{Add}}$
• in the *Custom Application Launcher* window, set
  Type to *Location*
  Name to a name that will be displayed when you mouse over the icon
  Location to the desired directory path: /home/USERID, /snd/USERID, /sflib or /sflib96
  COMMENT (optional) to a comment to display when you mouse over the icon

2.2. Manipulating desktop windows

Moving, switching between, resizing, closing and hiding windows
Often we need to move or to re-size windows on the monitor screen, or to make a different window the active window, in order to organize our working environment on the monitor. Follow these procedures:

☞ **To move a window to some other position on the desktop of the monitor screen:**
All systems: Move the mouse cursor into the title bar across the top of the window, click and hold the left mouse button, and drag the window to a new location.

☞ *To make a different window your* **active** window
All systems: simply click anywhere within the window that you wish to make active.

☞ *To re-size a window:*

All systems: Move the cursor into the lower right corner of the window until it changes shape (becoming highlighted and/or turning into an arrow shape). Then click and hold the left mouse button and drag it horizontally and/or vertically.
Full screen resizing, all systems: You can temporarily "explode" any window so that it covers the entire monitor screen (obscuring all other windows) by clicking on the *square box* icon, to the left of the $\boxed{\text{X}}$, near the upper right corner (Linux and Windows) or the upper right corner (Macintosh) of the window. To "downsize" this full screen window and return to a normal display of all windows, click on this icon again.

☞ *To close a window or quit the application:*

Sometimes our monitor screen will become cluttered with too many windows, whose applications also may be eating up too much RAM, and it is best close a window or to quit an application that we will not need for awhile.
All systems: Click in the **X** in the upper right corner (Linux and Windows) or upper left corner (Macinstosh) of the window. Alternatively, select *Quit, Exit* or *Close* from the *File* menu in the upper left corner of the window.

☞ *"Miniaturizing" (hiding) a window:*
Sometimes we may wish to leave an application loaded in RAM, so that we can return to it quickly, but temporarily "get this window out of the way" — reducing it to an icon or box on the taskbar or dock. To "hide" or "miniaturize" a window:
All systems: Click on the the third symbol, a boxed underscore, from the the right corner (Windows and Linux) or the left corner (Macintosh) of the window title bar.
The window will be reduced to a box and placed on the taskbar or on the monitor screen. If you log out, the next time you log in, this miniaturized window will reappear (and the application will be loaded into RAM).

A "hidden" application or window can be reactivated by clicking on its reduced window icon. While it often is convenient to "hide" an application window, remember that the application itself, and all of its data, still is loaded in RAM, and this reduces the amount of RAM available for your active programs. Thus, you should not overuse this technique, particularly when you are employing applications that may require a lot of RAM, such as soundfile mixing programs.

2.3. Directories (folders) and files

In most of your work you will be running programs or applications to create *files*. A *file* (often called a "*document*" on *Windows* and *Macintosh* platforms) may contain ascii text, or else data (such as digitized sounds or a digitized image) or else executable programs. Some programs enable us to create files, while other programs allow us to list or display these files, to change their names, to print them, or to remove them.

You can call a file anything you like, subject to the following limitations:

• The file name should not include any characters that may have a special meaning to Unix or (less often) to *Windows*. Examples of such "special" characters include ***** (an asterisk), **/** (a slash), \ (a back slash) and **#** (the pound sign).

• The file name should not include any blank spaces.

In Unix systems, the *space* character is treated as a *word delimiter* (separating strings of "words" or "arguments"). Files created with GUI-based applications on Windows and Macintosh systems <u>can</u> include blank spaces within file names. However, since you eventually may copy files created on one ECMC system to one or more of our other systems for use, you should make your file names cross-platform to avoid needless difficulties. The **.** ("dot") and **_** ("underscore") characters are often used in place of a *space* character in naming files, especially on Unix-based systems.

Related files, program applications and other system resources are organized on computer systems within "containers" known as **directories** on Unix systems and **folders** on Windows and Mac systems. (We will use the terms *directory* and *folder* interchangeably here.) A *directory* may include many files, and/or it may contain *subdirectories* (additional folders containing groups of related files).

☞ *Unix* systems employ a tree-like directory structure in which all directories branch out in "paths" from a "bottom level" system directory called *root* and abbreviated **/** ("slash"). The name of each directory within this path is preceded by a **/** (slash character).

☞ *Windows* file systems branch out from one or more "bottom level" folders corresponding to disks (or disk partitions) on the system. These disks or partitions begin with letter names, colons and backslashes, such as *C:\* (the primary system disk), and *D:\* (some other hard or removable disk or disk partition). The name of each folder within a path is preceded by a \ (backslash). ☞ The *Macintosh* file system is a mixture of Unix and of older (pre-*OS X*) hierarchies.

User home directories

When we create an account for a user on an ECMC computer system, we also create a **home directory** for the user — a folder in which the user can create and store files. The name of this directory is your *USERID* (your "user identification" name, by which you are known to the system and which you use to log on). You should have the same USERID on all ECMC systems on which you have an account (and, ideally, on your home system as well). This will simplify remote logins and file copying between ECMC machines and save you some typing.

Unfortunately, however, some commercial *Windows* (and a few *Macintosh*) applications permit read or write file access only to the user who installed the application. (These vendors apparently hope that each user will be forced to purchase and install her/his own copy of the application, a marketing ploy that we consider preposterous.) For this reason, on all ECMC Windows systems we currently employ a generic *ecmc* user account that is shared by all end-users. A separate account with administrative privileges is used by staff members for system maintenance. Although you must log on to the shared *ecmc* account — hardly an optimum situation from the standpoint either of user privacy or system security — you still will have an individual home folder, for your use only, on ECMC Windows systems.

☞ On the ECMC **Linux** systems, your home Unix directory has the path name

/home/USERID

This path name indicates that your home directory is a subdirectory within a primary system directory named *home* ("home directories of all users") that branches from the bottom (root) system directory directory indicated by the initial /.

☞ On **Macintosh** systems, your home folder has the path name

/users/USERID

☞ On ECMC **Windows** systems, your home folder will branch from the a *users'* disk partition whose name will begin with an alphabetical disk label, such as *E:* or *G:*
e.g. *E:\USERID* or else *G:\USERID*

Home soundfile directories

The ECMC *Linux* systems that are used for music production also include separate disks or disk partitions ("areas") in which users store binary *soundfiles*. This is done to optimize system performance for audio applications. For maximum throughput, soundfiles should be stored on separate disks in large contiguous blocks. Isolating soundfiles from other types of files helps minimize fragmentation when these soundfiles are written to disk. Thus, you actually have two "home" directories on these Linux systems:

(1) your "regular" Unix *home* folder (*/home/USERID*), for ascii files and other types of non-audio files; and
(2) your **home soundfile** directory.

Your home **soundfile** directory has the path: */snd/USERID*
This directory is an area on the *snd* ("sound") disk, which branches from the root directory.

☞ On ECMC *Windows* and *Macintosh* systems, user have only one home folder, in which they store both soundfiles and all other types of files. However, many ECMC users create a *sounds* folder within their home folder and store all soundfiles within this subdirectory.

2.4. Unix shells

One of the most basic programs on all Unix systems is an interactive **shell interpreter**, which opens a window into which you can type in (and thus execute) Unix *commands* recognized by the *shell* and receive program output and error messages from these commands. A *command* is the name of an executable program or application. The shell program itself includes many built-in programs and utility operations, and recognizes hundreds of other Unix commands and audio applications.

Before the advent of graphical window managers in the mid 1980s the shell was the only user interface on all Unix systems. When users logged onto a mainframe or mini-computer, a single full screen shell window opened on their monitor screen. Even today, when we remotely log on to an ECMC Unix system (either from another ECMC system or, via the internet or a modem, from any other computer), we often communicate with the system by means of a shell program. When a shell is ready to accept input from a user, it displays a *prompt* at the left edge of the current line in the window. The system administrator, or an advanced user, can set or change this prompt, which might be a dollar sign character (*$*), or your *userid*, or your current directory, or some other symbol or character string.

The music and general purpose software on the ECMC Unix-based systems includes both

(1) non-graphical Unix and audio software commands and programs, which are executed by typing in the name of the command in a shell window; and
(2) graphical (*GUI*) applications, which can be launched either by single clicking (Linux) or double clicking (Macintosh) on an icon for the application, or else by typing the name of the application in a shell window.

Some of our most important musical and audio operations (such as editing soundfiles) can only be performed with a *GUI* application, while others can be accomplished only, or more easily, by typing in *commands* and *arguments* (options to these commands) in a shell window. And some operations, such as playing a soundfile, can be accomplished either by means of shell commands or by opening a graphical application, whichever is handiest at the moment.

☞ **To open a shell window:**
☞ **Linux** : Single click on the Terminal icon. You probably have placed a *terminal* launcher icon on your taskbar panel. If not, click on *Applications->System Tools->Terminal* to open a shell window. Many ECMC users prefer to have two or more shells open at all times on their desktop. I often have three or four shells running simultaneously.
☞ **Macintosh** : Double click on a Terminal icon. This icon is located in *Applications/Utilities*. Drag a copy of the icon to the dock so that you can access it quickly.

Shell windows can be moved, re-sized, "hidden" and closed in the same manner (described above) as all other windows. You can change the font and font size, enable or disable multicolored display, and set various other preferences for the window, by popping open an options menu with the right mouse button.

Type of shells

Several variants of the shell program are available on Unix systems. The original shell program, written by Stephen Bourne in the early 1970s, is known as the *Bourne shell* and is abbreviated *sh* ("*sh*ell"). In the early 1980s versions of Unix distributed by the University of California, Berkeley, introduced an alternative, widely-adopted shell with a syntax similar to that of the programming language *C*, known as the *C-shell* and abbreviated *csh*. During the 1990s, updated versions of both the Bourne shell and C-shell were introduced, known as

- the *Bourne Again* shell : abbreviated *bash*, downwardly compatible with *sh*, preferred by most users and the default shell on Linux and Mac systems; and
- the *tcshell* : abbreviated *tcsh*, downwardly compatible with the *csh*, with some added features; the *tcshell* was the default shell on early versions of Macintosh *OSX*, but is used infrequently today

Although several shell variants remain available, *bash* is by far the most widely used shell today. To find out which shell you are using you can type:

echo $SHELL

All of these shell variants share certain basic features and commands, but differ in options, in certain syntactical conventions, and in how they define *environment variables* ("preferences" and alias abbreviations that often are exported by the shell to other programs and applications). The ECMC Turnkey package include many locally written *shell scripts* — executable programs written using a series of shell commands.[5]

2.4.1. Setting your system password

When we create an account for you on an ECMC system we will assign you an initial password. Whenever you log on to one of these systems, the operating system will prompt you to type in your *userid* and your password, in order to prevent someone else from trying to log on as you. As soon as possible, you should change the temporary password we have given you to a character string that only you will know. Use eight or more characters, and avoid obvious codes such as a birth date, someone's name, or anything that could be found in an online computer dictionary. Ideally, your password should be easy for you to remember, but very difficult for someone (or a program written by some computer hacker) to guess. Since all of our systems are connected to internet, we must take precautions to minimize the chances of hackers gaining access to these systems by cracking the password to some user account. Never give your password to another user or include it in a e-mail message or phone conversation, and change this password every few months.

Changing your password

☞ **Linux** systems: Type **passwd** in a shell window. You will be prompted to type in your old password, then your new password twice.

☞ **Macintosh** users also can use the *passwd* command in a shell to change their password. To use the *OSX* GUI tool instead, click on the Apple icon and select *System Preferences*. Select *Accounts*, click on the lock icon in the bottom left corner and enter your current password. Now select an account in the left panel and select *Change Password*. Enter your new password and click *Reset Password*.

☞ **Windows** : As noted above, currently, all non-administrative users on our Windows share a single login account, and need only know the current password for this account, which you will be given. Obviously you should not try to change the password on this shared account.

[5] ECMC shell scripts have been written and revised over a period of more than twenty-five years. The original versions of some of these scripts were written on Sun, NeXT and SGI systems on which the *csh* or *tcsh* was the default shell. A few of the older ECMC shell scripts are written as *csh* scripts, while all of our more recent ECMC utilities are written as *bash* scripts. However, this is transparent and unimportant to the end user.

2.4.2. Some elementary Unix commands : date, who and cal

The command **date** displays the current time and date on your terminal. To issue this command, simply type

```
date
```

in a shell window and you should see something like

```
Mon Jul 14 13:55:32 EDT 2003
```

This display is the *standard output* (sometimes abbreviated *stdout*) of the *date* program. By default, the standard output of *date* (and of most other Unix programs) is sent to the shell window from which you submitted the command.

The command **who** lists all users currently logged on to a computer.

(When a user has opened multiple shell windows, a separate login entry appears for each of these shells. Each of these windows is an independent process running a shell command interpreter program.)

To obtain a calendar for a particular year you can use the Unix program **cal**. Type *cal* followed by a space and then a year. For example, typing

```
cal  2013
```

will bring a calendar for the year 2013 to your monitor. Typing *cal 2025* would cause a calendar for the year 2025 to be displayed. Typing

```
cal 12  1900
```

would bring a calendar for December, 1900 to your screen.

cal is an example of an executable program that requires one or more **arguments** (additional information, or data, required to execute the program). In this case, the argument(s) include the particular year (and, optionally, month) for which we wish to obtain a calendar. While some simple programs, like *date* and *who*, require no arguments, other programs, like *cal*, require one or more arguments before they can be executed.

Most Unix programs allow us to **redirect** the *standard output* of a program so that instead of coming to our terminal monitor it is written to a file on the disk. The Unix *redirect* symbol is **>** (the "greater than," or "right arrow" symbol). Typing

```
date  >  currenttime
```

will write the date display into a file called *currenttime* in your current working Unix directory on the disk. Typing

```
cal  4 2050  >  temp
```

would write a calendar for April, 2050 into a file called *temp*. If a file named *temp* already exists in our directory, it will be overwritten. If you wish to append the output of a program to the current contents of a file, rather than replacing the current contents of this file, use the append redirect symbol

```
>>
```

Example: Typing

```
cal 2012  >>  temp
```

will add the output of the *cal* program to the end of the current contents of file *temp*.

Practice using these elementary Unix commands, and redirecting the output into new files or appending this output to the end of existing files. Create at least one fairly lengthy file of 40 or more lines. These scratch files will be useful to you in trying out some of the commands that follow.

2.5. Listing, viewing, copying, renaming and deleting Unix files

Listing your Unix files : ls

The command **ls** (short for "**l**i**s**t"), with no arguments, will list all of the files in your current working Unix directory. Typing

```
ls  filename    or   ls  file1  file2  file3
```

(where *filename, file1, file2* and *file3* are the names of existing files)

will list only the specified file(s). Typing **ls c*** will provide you with a listing only of files whose names begin with a lower case *c*. The ***** (asterisk) is a Unix "wild card" (or *metacharacter*) symbol that means "anything." In this case, we are asking *ls* to list all files that begin with the ascii character "c," followed by any other character string. Typing **ls *info** will list any files that end with the character string *info* (perhaps a file called *projectinfo* and another called *helpinfo*). Typing **ls *info*** would return the names of files that include the character string *info* anywhere within the file name. In addition to listing files with names such as *soundinfo* and *helpinfo*, this particular argument to the *ls* command would list files with names such as *newinformation*.

The *ls* command can also be used to give us a *long* listing of our current files, which includes information stored in the *header* of a file. (Some of this information is similar to that obtained in *Windows* when one clicks on a file's *Properties* tab.) To obtain a long listing of all of the files in our current working directory, we include a **-l** *flag* argument to the *ls* command:

ls -l

The output of this command will look something like the following:

```
(permissions)   (links)  (owner)  (size)   (date)           (name)

d rwx r-x r-x   2        allan     512     May 24 13:11     Instrs
d rwx r-x r-x   3        allan    512      Feb 17 10:30     SCORES
- rw- --- ---   1        allan      85     Jun 24  2001     file1
- rw- r-- ---   1        allan    107899   Jul 25  2002     file2
- rwx --x --x   1        allan     158     June 3 18:06     file3
```

The *name* field indicates the name of the file or directory. The *links* column rarely concerns us. The *date* field shows the time at which the file was created or last edited (altered). The *owner* and *size* fields indicate who owns the file and its size in bytes. Many users prefer to see file sizes displayed in "human readable" format, such as 105K (105 kB) rather than in raw bytes, such as 107899. The command *ls -lh* will produce a "human readable" size display. We have set the *-h* option as a default on some ECMC Unix systems.

Permissions field: There are 10 characters in the *permissions* field. The first column indicates whether the entry is a file or a complete subdirectory. If the first character is blank, indicated by a slash (**-**) the item is a single file; if a **d** appears in the first column (as in the *Instrs* and *SCORES* items above), the item is not a single file, but rather a complete subdirectory, which may include several files. To list the individual files (and any further subdirectories) within directory *Instrs*, type

ls Instrs or (for a long listing) *ls -l Instrs*

The other 9 characters in the permissions field indicate which system users have permission to
- **r** : *read* (look at) a file;
- **w** : *write to* (alter or remove) it; and,
- **x**, which can have one of two meanings:
  if the file is an executable program, to *execute* this program
  if the listing instead is a *directory*, to browse through and open files within this directory

---

☞ *More information for advanced users:* (Beginning users can skip the following small print discussions of *permissions*, if you wish, and jump ahead to *Viewing a file*.)

The last 9 characters are organized into blocks of three, which indicate, in order, read (*r*), write (*w*) and execute (*x*) permissions for the *owner* of the file, for other members of the owner's system *group*, and finally for all *other* system users:

owner	group	all others	(file name)
r w -	- - -	- - -	file1
r w -	r - -	- - -	file2
r w x	- - x	- - x	file3

In the listing above, *file1* can be read and altered by its owner; no one else (except a staff member who temporarily becomes the system *superuser*) can access this file. (Perhaps this is a love letter, and I don't want anyone messing around with it.) *file2* can be read by other members of my group, but only I (or the system *superuser*) can alter or remove it. *file3,* which appears to be an *executable* program I have written, can be run by anyone, but can only viewed or changed by its owner.

Using the Unix command *chmod* you can change the permissions of a file or directory.

Viewing a file: cat, more and **less**

To view an ascii file on your monitor screen, type

cat filename

cat may seem like a strange command name to use for viewing files. Actually, it is an abbreviation for *con*-**cat***enate* ("merge"), and the *cat* command will accept several arguments. Typing

cat file1 file2 file3

will bring all three of the specified files to our monitor. If these three files contain many lines of text, the display will scroll forward rapidly on our monitor before we have time to read the top lines. To freeze a display while it is scrolling, type $\boxed{\text{Control}}$ $\boxed{\text{s}}$ (hold down the *Control* key and type an *s*); to resume scrolling, type $\boxed{\text{Control}}$ $\boxed{\text{q}}$

If a long file goes whizzing by in our window, we always can use the window scroll bar to scroll back up in the window and take a look at previously displayed data. The scroll bars can be very useful for reviewing something we typed earlier, or for looking again at program output. However, sometimes this is inconvenient, and we simply wish to display data one screenful at a time. Like *cat*, the *paging* programs **more** and **less** also display ascii files, but bring only one "page" to the screen at a time. (The number of lines in a page, or screen, will depend on the size of the shell window and the font size.) To scroll down one page tap the *space* bar; to scroll forward by only a single line, hit a *carriage return*.

The program *less* allows somewhat greater flexibility in viewing files than *more*. (Bad pun: "*less* is *more*".) Additional scrolling commands in *less* (identical to some of the scrolling commands used with the Unix text editor *vi*) allow us to scroll backwards as well as forwards:

$\boxed{\text{Control}}$ $\boxed{\text{f}}$ scrolls **f**orward a full 25 line page (identical to tapping the space bar)
$\boxed{\text{Control}}$ $\boxed{\text{b}}$ (short for "**b**ackwards"), scrolls back a full page
$\boxed{\text{Control}}$ $\boxed{\text{d}}$ scrolls "**d**own" (forward) a half page
$\boxed{\text{Control}}$ $\boxed{\text{u}}$ ("**u**p") scrolls backwards a half page

To use *more* or *less*, type:

more filename or *less filename*

To view several files, type

cat file1 file2 file3 | *less* (or *more*)

The **|** symbol in this command line is a Unix **pipe** ("connect") sign, which means: take the output of the first program (in this case *cat*) and send it as the input to the following program (in this case *more* or *less*).

more terminates automatically and returns you to a shell prompt. *less* must be terminated by typing: **q** (for "**q**uit").

Two additional ascii file display utilities — **head** and **tail** — sometimes are handy:

Typing *head filename* displays only the beginning (first ten lines) of the file.
Typing *tail filename* displays only the end (last ten lines) of the file.
head -3 filename displays only the first three lines of the file
tail -15 filename displays the last 15 lines of the file
tail -22 file1 > file2 copies the last 22 lines of file *file1* into a new file called]fIfile2
head -6 filename | *tail -2* displays lines 5 and 6 of the file

Manipulating files: cp, mv and **rm**

• To make an exact copy of a file, use the Unix **cp** ("**c**o**p***y*") command, typing

cp sourcefilename newfilename

Example: Typing

cp systemstuff sysinfo

will copy the contents of file *systemstuff* into a new file named *sysinfo*.

• To change the *name* of a file (but not its contents), use the **mv** ("move") command:

mv oldname newname

(More on the *mv* command later.)

• To *delete* one or more files, use the **rm** ("**r**e**m***ove*") command:

rm filename

or

rm file1 file2 file3

Be careful when using *rm*. By default, you will not be prompted to confirm that you really do want to delete an existing file. And once a file has been deleted it cannot be recovered, unlike graphical *trash* bins that do not physically delete files until someone manually "empties the trash." (You can change this default behavior if you wish, so that the shell will prompt you for confirmation before deleting any file.)

Special wild card characters, such as the asterisk ("anything") symbol, can be used with *rm* and many other Unix commands:

*rm info**

will remove all files that begin with the character *info*. Be extremely careful, however, about using the ***** symbol in combination with a *rm* command. If we were inadvertently to put a space between the string *info* and the * in the command line above, like this

*rm info **

the results would be disastrous. *rm* would delete a file named *info* (if such a file exists), and then would delete all of the files in this directory, wiping out everything in this folder.
To prevent this type of blunder, some Linux distributions by default employ the *-i* ("interactive") option with commands such as *rm*, *mv* and *cp*, prompting the user for confirmation before deleting or overwriting any files. Advanced users often find this "interactive" option annoying and turn it off.

Hidden (dot) files and directories

When you use the *ls* command to list your files, you actually obtain only a partial listing. Your home Unix directory also includes several hidden ascii files and also hidden directories (containing several files) that begin with the character **.** (a dot, or period). Normally we do not want to list these files and directories, which include *preference* files and control various aspects of our Unix environment, but sometimes we *do* need to access this "hidden world." To list all of the Unix files in your home directory, type *ls* with the *-a* flag option:

ls -a or *ls -la*

By editing these dot files with a text editor, users can customize various Unix programs. Perhaps the most important of your dot files are *.bashrc* and *.bash_profile*, which set various options and preferences for your *bash* interpreter.[6] Many applications, such as *Firefox* and *ssh* and many audio applications contain entire dot directories with files that customize the application and store user preferences.

2.6. Online help and system documentation

A variety of online help and documentation programs are available on the ECMC systems, and can be accessed at any time you need information on a topic. This system documentation includes

(1) graphical help and tutorial documents on basic the *Linux*, *Macintosh* and *Windows* system resources bundled as part of the operating system distribution
(2) Unix *man* (manual) pages (Linux and Macintosh systems), available both for standard Unix commands (such as *cat* and *ls*) and, on Linux systems, for many of our non-graphical ECMC Turnkey music programs
(3) local (Eastman Computer Music Center) *ecmchelp* files (Linux systems)
(4) the *Docs* documentation section of the ECMC World Wide Web site; and
(5) online help documents available within many graphical music, audio and general purpose applications

(1) Graphical **help documents** bundled with the **operating system** (all systems)

I suggest that beginning users on a system take a few minutes of their first few work sessions system to explore these documents.

☞ **Linux** : Tutorial help on the *Gnome 3* desktop environment can be accessed by selecting *Applications->Accessories->Help*.

[6] The corresponding file for *tcsh* shells is *.cshrc*.

☞ **Windows** : From the Start menu select *Help*. In the window that opens, click on the *Contents* tab to view topics arranged by subject, or the *Index* tab to display a list of help topics, or else on the *Find* tab to search for information.

☞ **Macintosh** : Extensive help on many aspects of *Mac OSX* can be accessed by clicking on the *Help* menu on the top panel. Type in one or more words in the *Search* box field

(2) Unix **man** (manual) pages (Linux and Macintosh systems)

man pages are available for almost all general purpose Unix commands, and for many (but not all) command line music software programs submitted from a shell. *man* pages generally are not available for graphical applications (which generally provide their own graphical help utility). To bring a Unix manual page for some program to your screen, type

man commandname

from a shell prompt.

Examples:

man ls

will bring a manual page for the Unix program *ls* to your screen.

man findsflib

will display a manual page for the ECMC program *findsflib*, which can help you locate particular types of *sflib* soundfiles

The command **whatis** will tell you the function of a Unix program. For example, typing : *whatis cat*
will tell you what *cat* does. Note that *whatis* works only with standard Unix commands, not with ECMC programs such as *findsflib*.

On *Linux* systems, the command *info*, which can display *man* pages for Unix programs (but not music software) , as well as additional documentation on certain Linux programs, can be used in place of *man*. Typing

info cat

will display the *man* page for the *cat* program.

(3) *Online docs* section of the *ECMC web pages* (all systems)

The Center maintains a fairly extensive web site that can be accessed at the URL

http://www.ecmc.rochester.edu

The HTML documentation in these pages is accessed by our own users (as an in-house *intranet*) as well as by interested internet visitors who find this site useful. If your set this URL as your *home page* in the *Preferences* of your web browser on an ECMC machine such as *madking* or *gesualdo*, the ECMC home page will open automatically when you launch the browser. By clicking on the *Docs* link from the ECMC home page, or by navigating directly to

http://www.ecmc.rochester.edu/docs.html

you can display up-to-date documentation on our software and hardware resources for both studios. You should become familiar with the contents of this web page documentation so that you can access this information quickly whenever necessary.

(4) **ecmchelp** files (Linux systems)

We have created a series of ascii Eastman *help* files that cover various topics and procedures commonly used on our Unix (and, to a lesser degree, Windows) systems, and bugs or problems encountered in using some programs. These files vary somewhat from system to system. To obtain a list of the local help files currently available on the system on which you are working, simply type

ecmchelp

in a shell window. To read one of these available help files, type

ecmchelp topicname

These files are displayed one screenful at a time. Tap the space bar to see the next page, or type a **q** to quit. Copies of several of these files are included in the *Docs* pages of the ECMC web site.

(5) Graphical *help* within applications

Many music, audio and general purpose applications, such as soundfile editing and mixing programs, include graphical help documents that can be accessed while you are running the application. The format of the documentation may be similar to that of the online Windows, Linux or Macintosh help documents, or the application may open a web browser and display pages on the application on which you are working. In most cases you can open an application's online *help* by clicking on the word *Help* near the upper right corner of the window, then, in the menu that pops open, select the topic that you want to display.

2.7. Printing

The ECMC Linux, Windows and Macintosh computers all are connected to a networked Hewlett Packard model *8000N* laser printer in room 53 that is capable of printing at resolutions up to 1200 dpi ("dots per inch") on paper sizes up to 11 by 17 inches at speeds of up to 24 pages per minute. Please note that this printer is designed for printing comparatively short files of about ten or fewer pages, and is not intended for printing massive documents or large scores created with music notation programs. After printing, make sure that there is sufficient paper in the paper trays, and load additional paper if necessary. Please do this carefully and gently so as not to damage the plastic paper tray of the printer nor to jam paper into the feed mechanism. The *8000N* is a rather fragile behemoth and it is easy to jam paper if you are not careful.

To print pages from within a graphical application or web browser, simply click on the Print icon within the program, or else select *File* and then *Print* or type *ˆp* (hold the control key down while typing a *p*) and then follow the dialog box instructions.

Linux systems : To print out an ascii file from a shell window, use the generic Unix "line printer" command, typing:

lpr filename

To print out several files in one batch process, concatenate the files and pipe the output to the printer:

cat file1 file2 file3 | lpr

To print out a *man* page or an ECMC *help* file, pipe the output of the *ecmchelp* or *man* command to the printer:

ecmchelp pitchratios | lpr

If you experience any problems with the printer, immediately notify the system administrator by e-mail.

2.8. Text editors

A *text editor* is a program that allows us to type ascii text directly into a file and to edit this text, making corrections, additions and deletions. A *word processing* application such as Microsoft's *Word*, by contrast, includes many additional features (e.g. for formatting and fonts) that you will not need for your work in the ECMC studios.

☞ **Linux** systems : Most Unix distributions come bundled with many text editors, and you have a choice of using either a screen-oriented program, available on all Unix systems, or else a mouse-based GUI application. Different GUI editors are available on various Unix distributions, but most of these applications are very easy to learn and use and employ fairly standard *select-cut-paste* conventions. Linux distributions several simple GUI text editor choices, with names such as *gedit* (Gnome editor), *kedit* (KDE editor) and *nedit*. For simple tasks you can use the simple mouse-based editor *gedit*, or else the somewhat more powerful editor *nedit*, by typing

gedit & or else *nedit &*

in a shell window. Eventually, you may decide to use *gedit* to run *SuperCollider*.

Longer term users of the ECMC Unix systems are encouraged to learn a much more powerful screen-oriented editing program, bundled in all Unix/Linux/Mac distributions, known as *vim* (also known as "*vi*"). While certainly not among the simpler or "friendlier" text editors to learn, *vim* can be very powerful, and — once you learn a few of its conventions — much quicker than GUI editors for performing certain types of tasks, such as quickly making several substitutions. Links to tutorials on using *vim* are included in the on the *Docs* page of the ECMC Web pages. Another powerful text editor bundled with all Linux, Unix and Mac distributions is *emacs*.

For the preparation of major documents with formatting, or opening (and editing and saving) documents in Microsoft *Word* format, the word processor bundled within the *LibreOffice* suite is recommended. The *LibreOffice* word processor can read and write files in several formats, including Microsoft *Word .doc*, and Open Document *.odt* in addition to *ascii*. At some point you might also find the spreadsheet or other components within *LibreOffice* useful.

☞ **Windows** systems: On the ECMC *Windows* systems your choices for text editing are the bare-boned *Notepad* application, the slightly more sophisticated *Wordpad* application, which can write and read files in Microsoft *Word* format, and *LibreOffice*. You should be careful in which format you save files with these applications. If you will be using the file with another application, save the file as a generic ascii ("text format") file with line breaks, although this will delete some formatting from the file. For those familiar with the open source *vim* text editor, a graphical *Windows* version called *gvim* is available on all ECMC *Windows* systems.

☞ **Macintosh** systems : The simple Mac GUI text editor *TextEdit* may meet your text editing needs for basic tasks. Somewhat more powerful graphical text editors also are available, *vim* and *emacs* can be run in a shell window, and *LibreOffice* is available. HERE HERE

2.9. Some additional Unix resources

Hundreds of utility programs come bundled with Unix distributions — spell checkers (such as *ispell* and *aspell*), file compression and uncompression programs, and so on. Two utilities that many ECMC users often find useful are *grep* and *diff*.

grep is an example of a *filter* program. *grep* displays all occurrences of a specified ascii character string that occur within one or more specified files. The syntax is

grep characterstring filename(s)

Examples:

```
       string   file(s)
grep   oboe     woodwinds
```

Result: All lines in file *woodwinds* that include the character string *oboe* are displayed on your monitor screen.

*grep ob **

Result: All lines containing the character string *ob* that occur within any file in your current working directory are displayed.

grep -e string1 -e string2 filename

Result: all instances of character strings *string1* and *string2* within file *filename* are displayed.

The command **diff** compares two similar files (most often a source file and a copy that has been edited) and displays all lines that differ between these two files. This can be helpful when comparing a newer and older version of the same file. The command syntax is

diff file1 file2

2.9.1. Command history

The shell program that interprets your Unix commands includes several easy-to-use features that can simplify your work. For example, your *bash* (or *tcsh*) shell keeps a record of the commands you have typed. To view this command history, type

history

(or simply the ECMC alias abbreviation **h**) and you should see something like this:

```
1  date
2  ls
3  cat  file1  file2  file3  >  bigfile
4  vi  bigfile
5  play  soundfile1   soundfile2  soundfile3
```

• To repeat the last command you typed (in this case, the command to play three soundfiles), type

!!

• To redo an earlier command, type **!***commandnumber*. For example, to redo command number 3 above (concatenating three files and sending the merged output to a new file), type

!3

• If you mistype a command, you can redo it, using the substitution symbol ˆ to make corrections. For example, if we wanted to play four soundfiles, but clumsily type *plau* instead of *play*, like this

plau soundfile1 soundfile2 soundfile3 soundfile4

we need not retype the whole command line, but simply

ˆauˆay

followed by a carriage return. The corrected command

play soundfile1 soundfile2 soundfile3 soundfile4

will cause the four soundfiles to be played.

To cycle backwards and forwards through recent shell commands you can use the down and up arrow keys to the left of the numeric keypad (and directly under the *Delete* key). Once you have located a previous command that you would like to repeat, simply tap a carriage return. To modify this command, use the <- left arrow and -> right arrow keys to move the cursor one character at a time to the left or right within this command line. Unwanted characters can be removed with the $\boxed{\text{Delete}}$ key, and new characters can be inserted at the cursor insert point simply by typing them in.

Filename completion

The *bash* shell provides filename completion, which works with many basic Unix commands. If you have a file with a long or complicated name, such as *longviolinsolo*, that you wish to look at, try typing something like

cat longv

followed by the $\boxed{\text{tab}}$ key. If the string *longv* matches only this one file, your shell interpreter will complete the file name for you and then execute the command. If the string matches more than one file name, the shell will present you with alternatives. Try using filename completion with commands such as *ls*, *less* and *more*.

Filename completion also works with many command names. If you type the first few characters of a command followed by a tab your shell will attempt to complete the command name.

Copying and pasting text

The *x11* windowing system incorporated into Linux includes a copy-and-paste buffer ("clipboard") through which we can copy and paste text between any two points within any open windows. If you learn how to use this buffer, you can save yourself some typing:

• To copy text into the buffer, drag over the text with the left mouse button until it is selected
• Move to the window into which you wish to copy the text, then click the middle mouse button to insert (paste) the text. This technique works with most GNU/Linux system programs, including text editors (*vi, nedit, emacs, kedit, gedit,* etc.) and most GUI applications, such as *Firefox* and soundfile editors.

2.9.2. Creating directories

When you first begin working on a computer system you will not have many files, and may be content to keep all of your *ascii* files in your home Unix directory and all of your soundfiles in your home soundfile directory. With time, however, you will accumulate ever more files, and at some point will want to organize these files by moving most of them into subdirectories. For example, you might wish to keep all of the files you use to create one section of a composition within a single subdirectory, and call this subdirectory something like *Section1* or *SEC1*. Many ECMC users prefer to use all capital letters when naming directories, or to begin directory names with a capital letter, while employing all lower case for file names, so that when listing the contents of a directory, subdirectories (folders) containing additional files will be visually more obvious.

On Unix-based systems you can create directories, and move, copy and delete files, either by means of the graphical Gnome or Macintosh *Finder* file browsers or else in a shell window. We will concentrate here on shell commands.

• To make a new subdirectory within your current working directory, type

mkdir directoryname

(where *directoryname* is the name you wish to give to the folder).

• To move some files into this subdirectory, use the *mv* command, which we have used so far only to change the names of files. *mv* can also be used, however, literally to *move* one or more files from your current directory to some other directory. To use *mv* in this fashion, include the names of all the files you wish to move to some subdirectory, and then, at the end of the command line, include the target subdirectory name:

mv filename(s) directoryname

Example: *mv file1 file2 file3 Mixes*

will move the three named files into your subdirectory *Mixes*.

mv piano Notes*

will move all files in our current directory that begin with the character string *piano* into our subdirectory *Notes*

• To move yourself into one of your subdirectories (that is, to make this subdirectory your current working Unix directory), type

cd directoryname

• To find out what directory you are in at any time, type **pwd** (short for "**p***rint* **w***orking* **d***irectory*").

• To move back into your home directory, type **cd**. (By default, with no argument specified, *cd* moves you back into your home Unix directory.)

The ˜ (tilde) symbol is a Unix shorthand for your home directory. Typing

mv file1 file2 file3 ˜

from any subdirectory will move these files to your home directory.

• If you wish to delete an entire directory, and all of its files, type

rm -rf directoryname (Are you sure you want to do this?)

• To delete an empty directory, type

rmdir directoryname

NOte that the *rm* and *rmdir* commands do not simply move files and folders into the *Trash*, where they can be recovered, but instead permanently delete the item(s) from the hard disk.

2.9.3. Job control

You can type in a series of Unix commands on a single command line by separating the individual commands with semicolons:

date ; ls -l ; pwd

In this example, the three commands would be executed consecutively.

Background jobs; killing jobs

The Unix commands you have run so far have likely executed very quickly, returning you to a shell prompt within a few seconds. Some processes, such as using a computer to synthesize music, however, can require longer to compute. We may wish to run such heavy duty crunching jobs in the background so as not to tie up a shell window, and so we can do other things while the process is running. A job placed in the background continues to run (although at a somewhat slower system priority), but allows us to use our shell window to do other things. We can place a job in the background by including an **&** (ampersand) character at the end of a command line.

Example:

*grep violin * &*

Result: The program *grep* will search through all files within your current working Unix directory for the character string *violin* and will display all matches. In the mean time, we can use shell window for other tasks. One disadvantage here is that the output of *grep* may come to our terminal at a bad time, garbaging up the display of the current process on which we are working. To prevent this, we can redirect the output of *grep* into a file:

*grep violin * > output &*

Now we can type away undisturbed and, at some later time, look at the file (here called *output*) that contains the character string matches found by *grep*.

Occasionally we may submit a job in the foreground, not realizing how long it will take to to complete, then wish to regain control of the window or terminal on which the sluggish job is running. We could kill the dilatory job by typing

$\boxed{\text{control}}\ \boxed{\text{c}}$

but some or all of the work completed so far on the job would be lost. Alternatively, we can *suspend* (temporarily halt, but not kill) the slow-running job by typing

$\boxed{\text{control}}\ \boxed{\text{z}}$

The computer will suspend work on this job and return us to our shell prompt, enabling us to do other work.

If we type the command: *jobs*
in this shell window, the shell will return a list of all jobs, whether active or stopped, that we are currently running in this window. The command *ps* will also list our jobs, along with the *process number* of each. To see a listing of all processes currently running on the computer (including system processes and processes of other users), type

ps aux

Eventually we can do one of three things with a stopped job:

- Resume its operation, by typing **fg** ("foreground") in its shell window
- Place the job in the background, so that it will resume operation but leave us with control of our terminal or window. To place a stopped job in the background and resume its execution type **bg**.
(The command *fg* will bring a job currently running in the background to the foreground.)
- *Kill* the job, by typing either: **kill -9 processnumber** or else
killall processname (where *processname* is the name of the command or program)

  Example: The command *killall audacity*
  will terminate all processes (and open windows) spawned by the *audacity* soundfile editor.

Find out the *process number* by using the *ps* command. For processes that you submitted from another window, including GUI apps, use *ps aux* . Note, however, that some complex processes create additional subprocesses, called *forks* or *children*, which usually (but not always) have consecutive *process numbers*. You must recognize and kill all of these *child* processes or you may leave orphaned zombie processes in RAM that are waiting for input that never comes.

If a Linux system computer ever seems to be responding unusually slowly to you, type *ps aux* to see if either you or some other user is running, or has left running, a CPU intensive job. Never submit heavy crunching jobs, or leave such jobs running in the background, while someone else is working in the sound room.

2.9.4. Batch jobs

If we wish to run a series of jobs in succession — that is, to issue a whole *batch* of jobs for consecutive execution — it is often easiest to type this list of jobs into a file and then execute the file. A batch job file might look something like this:

```
ls * >   longfile
grep  -e piano  -e violin longfile
echo "Here are my current mix files:"
ls  -lt  *mix*
du -h /snd/allan
```

If we type these lines into a file that we call *jobs*, we have created a Unix *shell script* — a series of commands to be executed by a temporary new "child" Unix *shell* interpreter. (Admittedly the example above comprises a rather inane sequence of random commands.) Now we can run the jobs in this file by typing:

bash jobs

(Actually, in a batch file this elementary, that uses only basic commands recognized by all shell variants, it really doesn't matter whether we use a bash (*bash*) shell or a simple Bourne (*sh*) shell to execute the batch file.)

Alternatively, we could mark our *jobs* file as executable by typing

chmod +x jobs

and then execute this file by typing

./jobs

(The *./* is an abbreviation for your current working Unix directory. This command instructs our current shell to create a temporary new "child" shell interpreter process to execute the file named "*jobs*" that is located in your current working directory.)

To run our script in the background and capture the *standard output* of the all of the jobs run into a file called *output*, type

```
bash  jobs  > output &
```

Many programs create not only a **standard output** (the data produced by the program, such as sound samples), but also a **standard error** output (abbreviated *sterr*). The *sterr* includes error messages, which can be helpful in determining the problem in case the program aborts, as well as diagnostic information that may be useful even when the program executes successfully. To capture both the *standard output* and any *sterr* messages, use the redirect symbol **>&**

```
bash  jobs  >&  output &
```

Some programs will not allow us to redirect the *standard output*. Many of these programs are music applications, like the software synthesizer *Csound*, that must write the computed samples (the *standard out*) into a soundfile on the */snd* disk. (These binary samples would be worthless to us if redirected into an ascii text file.) The *sterr* of programs such as *Csound* — what you see in a terminal window when you run a Csound job — includes information the soundfile it creates. When submitting a background jobs file that involves soundfiles, therefore, always use the **>&** redirect symbol.

One potential problem with shell scripts is that the "child" shell process they create does not inherit all of the current environment variables, aliases and functions of the parent shell from which you submit the job. The "child" shell will not know the current working directory of the "parent" shell, or its current working soundfile directory, and thus may write its output to the wrong directories, or be unable to find source files needed for execution. Many of the ECMC *Turnkey* utilities and aliases we employ to simplify running audio jobs on *madking*, such as *cdsf, lsf, mvsf, play* and *sfinfo*, are defined in a file called */etc/profile.d/turnkey.sh* . When you open the *terminal* application, the shell that is created *sources* (reads and executes) this *turnkey.sh* file, and all of the environmental variables, definitions, functions and abbreviations within this *turnkey.sh* file are added to the system environment variables "known by" all terminal shells.

To enable "child" shells to inherit all of a parent shell's environment variables, including these ECMC definitions, include this line at the beginning of a shell script:

```
source /usr/local/lib/ecmcbashsource.sh
```

The *ecmcbashsource.sh* file includes all of the important variable definitions and aliases from the *turnkey.sh* file.[7]

A batch file of *Csound* jobs might look something like this:

```
source /usr/local/lib/ecmcbashsource.sh
mko  MARIMBA  ;  score11 marimbatest1
csound  -d -o  marimbatest1.wav  orc  sout
score11  marimbatest2 ;  csound  -d -o  marimbatest2  orc  sout
score11  marimbatest3 ;  csound  -d -o  marimbatest3  orc  sout
sfinfo -s marimbatest*
```

Notice that we have included the *-d* flag to the *Csound* commands on lines 3-5 above to suppress displays of sine waves and other functions, and which require that the user close the displays before each Csound job terminates.

To run this file, perhaps called *marimbajobs*, in the background, and capture the diagnostic *sterr* output of Csound and the output of *sfinfo* into a file called *output* in your current working Unix directory, type

bash marimbajobs >& output &   or else   *sh marimbajobs >& output &*

At any time you can check on the progress of this series of jobs by viewing the *sterr* file output:

cat output   or   *less output*

Be sure that this batch job file has completed execution before you log off and leave the studio. If not, kill

[7] If you have created some of your own shell environment variables, alises of functions in your *~/.bashrc* preference file, you may want to have your script source this file as well:

source ~/.bashrc

all of the processes that are still running.

Here is another example that employs a *for* loop. I have created a Unix subdirectory named *Compositions* where I place all of my source files (e.g. *score11* input files) for my serious compositional work on *madking*. I want all of the source files for the composition on which I currently am working to be located in a subdirectory of *Compositions* named *Project2*. Within this subdirectory I have created eight similar test scores for ECMC library instrument *gran*, each containing a different *reseed* value that will affect all p-fields employing random number generation, and have called these score files *grantest1*, *grantest2* and so on through *grantest8*. I want all of the sound files for this composition to be consolidated within a single *snd* subdirectory that also is named *Project2*.

To create a soundfile from each of these test score files I could create a batch job file, named *jobs*, that looks like this:

```
#!/bin/bash                                   1
source /usr/local/lib/ecmcbashsource.sh       2
cd ˜/Compositions/Project2                    3
cdsf /snd/allan/Project2                      4
mko gran                                      5
for x in ‘ls grantest*‘ ; do                  6
    score11 $x; cs -d -o $x.wav orc sout      7
done                                          8
```

The line numbers above are included here only for convenience, and would not be included in my actual batch file. Line 1 (*#!/bin/bash*) tells the shell to use a bash interpreter. After insuring that the shell that will run the script knows all of my current environment variables (line 2), I tell the shell to "move" into the directory where the 8 "grantest" files are located (line 3), and also tell it to write all output soundfiles to my */snd* subdirectory */snd/allan/Project2* (line 4).

Now I am ready to gets some work done. Line 5 creates a Csound orchestra file. The *for* loop on lines 6 through 8 is executed eight times, for files *grantest1* through *grantest8*. The first time through the loop the variable *x* is set to *grantest1*, the second time through the loop it is set to *grantest2*, and so on through *grantest8*. The *x* is an arbitrary designation for this variable; we could instead have named the variable "z" or "*score*" or "*SCOREFILE*". Each of these score files is run through *score11*, then the resulting *sout* file and the orchestra file created on line 2 are used to compile an output soundfile whose name will be the name of the input score file plus the extension *.wav*.

To execute this *shell script* I could mark file *jobs* executable:

chmod +x jobs

and then run the script: *./jobs >& output*
If I don't want to bother marking the file executable, I could simply type:

bash jobs >& output

2.9.5. Disk usage

Disk space is not unlimited, and it is essential that all users monitor and control their disk usage on each system on which they have an account. This is especially important for soundfile storage.

- To find out how much space remains on various disks and disk partitions on a system, type **df** or **df -h** (the Linux -h option displays the data in "human readable" format)

- To check the available space for only the *soundfile* disk on a system in kilobytes, type
  df /snd or *df -h /snd*

- To check your own disk usage on the *sound disk* type
  du /snd/USERID or *du -h /snd/USERID* or (to get only a grand total) *du -sh /snd/USERID*
  where *USERID* is your login name.

- To check your current *Unix* file disk usage, move into your home Unix directory and type
  du or *du -h*

If your disk usage on your home directory or (more likely) your */snd* directory becomes excessive you may receive a testy e-mail message from the system administrator, or from a program we run periodically to check these things, advising you to slim down immediately. If you fail to reduce your disk usage after receiving such a message, we will do it for you, possibly (though not deliberately) removing your most important or beloved files. If you will not be working in the studio for several days, archive large soundfiles onto a flash drive or onto *cd* or *dvd* discs and delete them. Do not leave large soundfiles of completed works (for example, a finished semester project) on the disk for more than a few days.

Large production runs sometimes do require massive amounts of disk space — particularly multichannel works at high sampling rates. Advise the system administrator a few days in advance if you wish to balloon up in disk usage and we will try to accommodate you. Other users may be asked to delete soundfiles. You, too, may someday receive such a request. Please respond promptly, so that we don't have to zap some of your files.