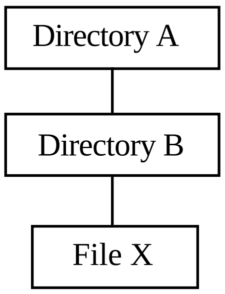
GRAPHICS AND TEXT – A COMPARISON

To present a full comparison of graphical and textual interfaces is far beyond the scope of these notes; many people have spent a long time thinking about the question and carrying out measurements of various sorts to find out just how the different sorts of interface work. We shall content ourselves with an example, which brings out some of the features of interest.

The example is contrived to illustrate certain features (it began life as an examination question, though it's evolved a bit), but it's realistic to the extent that the instructions work and perform certain operations on different sorts of operating system.

Consider this structure of directories A and B, and the file X



That directory structure could exist in any of the Unix, Macintosh, and MS-DOS (and therefore MS Windows) file systems, and in many more for that matter; they share a very similar structure, which is very widely used because it seems to do pretty well what we want. (We'll say more about this later, in the chapter *FILES IN THE SYSTEM.*) Suppose that the working directory is initially Directory A; in a GUI system, Directory A's window is open. Now we suppose that a sequence of instructions is executed; they are listed in their Unix form in the left-hand column of the table below. The other columns show the meaning of the Unix instruction, the corresponding meaning in a GUI system, one way to do it in a GUI system (usually a Macintosh way; there's usually something quite similar in the Windows 3.? File Manager); and a comment. There are some further comments (mainly on the comments) after the table.

	What it means in Unix :	What it means in GUI :	The instruction in GUI :	Comment :
cd B	Change the working directory from whatever it was (A) to B.	Change the active window from whatever it was (A) to B.	Double-click on the B icon in the active (A) window.	The system must determine the identity of the desired new working directory, and the fact that a change in working directory is required. To identify the directory in the GUI system, the system must be able to interpret the position of the pointer when the selection operation is received, so it must keep some sort of map of the screen. To identify the instruction, the operation (double-click, etc.) must be interpreted in terms of the context, particularly the system's information about the selected item..

cp X Y	Make a copy of the file X and call it Y.	Make a copy of the file X and call it Y.	There's no common "pure" GUI method; the closest is to select the file, then select some useful operation from an appropriate menu. Then you give the new file name in some way appropriate to the operation you selected.	The system must identify the original item, determine that a copy operation is requested, and get the new file name. The original item and request can be handled in much the same way as for the previous instruction. The new name requires more, because it isn't already there, so can't easily be made visible, and must be entered at the keyboard. (Neither Macintosh nor Windows provides a simple copy instruction, presumably because neither Apple nor Microsoft can think of a neat way to get the new file name. Windows provides copy as a menu item, and gets the new name through a dialogue box; in the Macintosh system, there is no explicit copy, and you have to make do by first duplicating the file then changing its name.)
mv X ..	Find the file now known as X in the working directory and list it instead as X in the directory ".." (a Unix convention to denote the parent directory, now B).	Cause the file X in the B window to appear in the window of B's parent.	Make sure that you can see both the A and B windows; then select the X icon in the B window and drag it into the A window.	The system must identify the instruction, the object to be moved, and the destination. For once, it's easier for the GUI, given the machinery we've already described. Provided that it can interpret the position of the pointer and has access to the properties of the entities represented on the screen, the task is fairly straightforward. In Unix, though, the mv instruction is ambiguous. In this example, the second name is that of a directory, and the interpretation given to the left is correct; but if it had not been a directory name, the file would have been renamed rather than moved. In the Windows file manager, the parent of an open directory is shown (curiously enough, as ..), so a direct equivalent of the text instruction is available.

mv Z ..	Find the file now known as Z in the working directory and list it instead as Z in the parent directory.	Cause the file Z in the B window to appear in the window of B's parent.	Can't be done – there's no Z icon in the active window.	Score a point for the GUI : you can't issue an instruction about object Z unless you can find it, and you can't find it if it isn't there.
rm *	Delete all the files (not directories) in the working directory.	Delete all the files (not windows) in the active window.	Select the files and drag their icons to "Trash".	The system requires the nature of the instruction, and the selection of operands. The delete instruction is fairly easy – drag to "Trash" on the Macintosh, delete key with MS-DOS or Windows, menu alternative for all – but it doesn't distinguish between files and directories. Selection is therefore necessary. The selection is harder; selecting all entries in a directory is reasonably easy, but selecting only files isn't. In the general case, you would probably have to pick them out one by one. Group selections can be built up item by item by controlling the context of the mouse operations; a click on an item while a specified key (shift on Macintosh, control in Windows) adds (or removes) an item to (or from) a selected group. (In the example, it's easy, because it happens that there's only one file left in B, but that's accidental.)

COMMENTS.

The most important thing to say about this example is probably that the real question is how the operating systems use their different interfaces to do what is just the same job in all cases. As we pointed out, the file systems themselves are essentially identical.

The big difference is that in a textual interface all the information needed must usually be provided explicitly in the instruction, while with the graphical interface there is much more cooperation between system and person. The system presents a lot of information on the screen, and all you have to do is select it. That makes life a lot easier for you, but requires a much more elaborate system.

It would be a mistake, though, to suppose that only the graphical systems give help in this way. There are two instances in the example where the textual system works the same trick. The first is the use of ".." to denote the parent directory; there's no direct equivalent with the Macintosh system (though there's a "shortcut" key combination which will find the parent window for you – you could argue that it's similar to the ..

convention in that you have to remember it), though the Windows File Manager does give you a special menu item (called ".." with an evocative up-arrow) to which you can drag your selected items. The second example is the "wild" character "*", used to mean "anything". That's a very simple example of a very powerful set of conventions with which you can define function of text called *regular expressions*. The nice thing about text is that it's very precise, and you can do precise operations on it; you can't do that with pictures.