NAME

twander – File Browser

OVERVIEW

Wander around a filesystem executing commands of your choice on selected files and directories. If you're new to ´twander´ and want to know why this program is better and different than whatever you're using at the moment, take a moment to read the section called **DESIGN PHILOSOPHY** toward the end of this document first.

Similarly, if this is the first time you've worked with ´twander´, there is a section towards the end of this document entitled **INSTALLING ´twander´** which describes how the program should be installed.

SYNOPSIS

twander [-bcdfhnqrstvwxy] [startdir]

OPTIONS

startdir

Directory in which to begin. (default: ./)

If this directory does not exist or cannot be opened, ´twander´ will display an error message and abort.

-b backcolor

Desired background color. (default: black)

-c path/name of configuration file

Specify the location and name of the configuration file. (default is ~/.twander)

If this file does not exist or cannot be opened, ´twander´ will display a warning to that effect but continue to run. This is reasonable behavior because ´twander´ provides a command to reload the configuration file without exiting the program (which you would presumably do after fixing the configuration file problem).

-d Start in debug mode. (default: debug off)

The program runs, but does not actually execute any commands. Instead, the contents of various internal tables such as the Symbol Table and Command Table are listed on standard output. The Key Bindings are also listed. If the user presses a defined Command Key, the command that would have been executed is printed to standard output, but no command is actually performed. This option is mildly useful in debugging configuration files insfar as it will display the Command String after all variable substitution (User-Defined, Environment, and Built-Ins) has been done.

-f forecolor

Desired foreground color. (default: green)

-h Print help information on stdout.

-n fontname

Name of desired font family. (e.g., courier, times, helvetica) (default: courier)

-q Quiet mode - suppresses warnings. (default: warnings on)

-r Turn off automatic refreshing of directory display. (default: refresh on)

Normally ´twander´ re-reads and displays the current directory every few seconds to reflect any changes that might have occurred to that directory's contents. This option is useful on slow machines (or slow X connections) and/or when working with very large directories. In this situation, the frequent updating of the ´twander´ display can make the program unacceptably slow and unresponsive. In this case you can still force an update manually with the REFRESH function (default assignment is to the Control-l key).

-s fontsize
Font size in points. (default: 12)

-t Turn off quoting when substituting built-in variables. (default: quoting on)

Anytime ´twander´ encounters a reference to one of the built-in variables (DIR, DSELECTION, DSELECTIONS, PROMPT:, SELECTION, SELECTIONS) in a command, it will replace them with **double quoted** strings. This is necessary because any of these can return values which have embedded spaces in them. By quoting them, they can be passed to a command or script as a single item. The -t option disables this behavior and replaces the built-in variable with unquoted literals.

-v Print detailed version information.

-w fontweight
One of: bold, italic, underline, overstrike. (default: bold)

-x width
Set window width. (default: 90)

-y width
Set window width. (default: 25)

KEYBOARD USE

By design, ´twander´ allows you to do almost everything of interest using only the keyboard. Various ´twander´ features are thus associated with particular keystrokes which are described below. It is also very simple to change the default key assignments with entries in the configuration file, also described below.

A NOTE ON KEYBOARD ARROW AND KEYPAD BEHAVIOR

Generally, the arrow and keypad keys should do what you would expect on the system in question. On Win32 systems, particularly, there ought to be no odd arrow/keypad behavior.

X-Windows is somewhat more problematic in this area. Just what an arrow key is "supposed" to do depends on how it's been mapped in your X server software. Testing ´twander´ on various X servers showed quite a bit of variability in how they handled the arrows and keypad. So ... if you're running in an X Windows universe and arrows or keypad do nothing, or do strange things, look into your key maps, don't blame ´twander´.

DEFAULT KEYBOARD AND MOUSE BINDINGS

Here, ordered by category, are the default keyboard and mouse bindings for ´twander´. The general format is:

Description (Program Function Name)
Default Key Assignment
Default Mouse Assignment (if any)

The "Program Function Name" is the internal variable ´twander´ uses to associate a particular feature with a particular keystroke or mouse action. You can ignore it unless you intend to override the default assignments. This use is described below in the section entitled, **CHANGING KEYBOARD BINDINGS.**

GENERAL PROGRAM COMMANDS

This family of commands controls the operation of ´twander´ itself.

Quit Program (QUITPROG)
Control-q

Exit the program.

Re-Read Configuration File (READCONF)
Control-r

Re-read the configuration file. This allows you to edit the configuration file while ´twander´ is running and then read your changes in without having to exit the program. This is handy when editing or changing command definitions. However, if you edit the configuration file and introduce an error, ´twander´ will terminate when you try to re-read it (just as it will if you try to start the program with a bad configuration file).

Refresh Display (REFRESH)
Control-l

Re-read the current directory's contents and display it. This is most useful if you have turned off automatic directory refreshing with the -r command line flag.

Toggle Details (TOGDETAIL)
Control-t

Toggle between detailed and filename-only views of the directory.

DIRECTORY NAVIGATION

This family of commands controls movement between directories. If at any point, you attempt to navigate into a directory that does not exist or which does not have appropriate permissions, ´twander´ will issue an appropriate message, and remain in the original directory where the request was issued. This is **unlike** the case of a non-existent or unreadable directory specified when the program is first started. In that case, ´twander´ reports the error and aborts.

Change Directory (CHANGEDIR)
Control-x

This is a shortcut that allows you to directly move to a new directory/path - i.e., Without having to navigate to it.

Go To Home Directory (DIRHOME)
Control-h

If the "HOME" environment variable is defined on your system, this will move you to that directory. If the "HOME" environment variable is not defined, this command will move to the original starting directory.

Go Back One Directory (DIRBACK and MOUSEBACK)
Control-b
Control-DoubleClick-Left-Mouse-Button

´twander´ keeps track of every directory visited and the order in which they are visited. This command allows you to move back successively until you get to the directory in which you started. This feature is implemented as a stack - each "backing up" removes the directory name from the visited list. The "Directory" menu (see **MENU OPTIONS** below) implements a similar feature in a different way and keeps track of all directories visited regardless of order, never discarding any entry.

Go To Root Directory (DIRROOT)
Control-j

Go to the root directory.

Go To Starting Directory (DIRSTART)
Control-s

Go back to the original directory in which ´twander´ was started.

Go Up To Parent Directory (DIRUP and MOUSEUP)
Control-u
Control-DoubleClick-Right-Mouse-Button

Move to the parent of the current directory ("..").

SELECTION KEYS

This family of commands controls the selection of one or more (or no) items in the current directory.

Select All Items (SELALL)
Control-Comma

Select every item in the current directory. The ".." entry at the top of the directory listing is not included. (We almost never want to include the parent directory when issuing a command on "everything in this directory". If you do wish to include the "..", do the SELALL command first, then click on ".." while holding down the Control key.)

Invert Current Selection (SELINV)
Control-i

Unselects everything which was selected and selects everything which was not. As with SELALL, and for the same reason, the ".." entry is never selected on an inversion.

Unselect All Items (SELNONE)
Control-Period

Unselect everything in the current directory.

Select Next Item (SELNEXT)
Control-n

Select next item down in the directory.

Select Previous Item (SELPREV)
Control-p

Select previous item up in the directory.

Select Last Item (SELEND)
Control-e

Select last item in the directory.

Select First Item (SELTOP)
Control-a

Select first item in the directory. This will always be the ".." entry, but it is a quick way to get to the first part of a very long directory listing which does not all fit on-screen.

The mouse can also be used to select one or more items. A single-click of the left mouse button selects a particular item. Clicking and dragging selects an adjacent group of items. Clicking an item and then clicking a second item while holding down the "Shift" key also selects an adjacent group of items. Finally, a group non-adjacent items can also be selected. The first item is selected with a single left mouse button click as usual. Each subsequent (non-adjacent) item is then selected by holding down the "Control" key when clicking on the item.

SCROLLING COMMANDS

If a given directory's contents cannot be displayed on a single screen, ´twander´ supports both vertical and horizontal scrolling via scrollbars. This capability is doubled on the keyboard with:

Scroll Page Down (PGDN)
Control-v

Scroll down one page in the directory listing.

Scroll Page Up (PGUP)
Control-c

Scroll up one page in the directory listing.

Scroll Page Right (PGRT)
Control-g

Scroll to the right one page width.

Scroll Page Left (PGLFT)
Control-f

Scroll to the left one page width.

COMMAND EXECUTION OPTIONS

This family of commands causes ´twander´ to actually attempt to execute some command you've chosen:

Run Arbitrary Command (RUNCMD)
Control-z

This is a shortcut that allows you to run any command you'd like without having to define it ahead of time in the configuration file. It is more-or-less like having a miniature command line environment at your disposal.

Run Selected File / Move To Selected Directory (SELKEY and SELMOUSE)
Return (Enter Key)
DoubleClick-Left-Mouse-Button

If the selected item is a Directory, ´twander´ will move into that directory when this command is issued. If the selected item is a file, ´twander´ will attempt to execute it. Whether or not the file is actually executed depends on how the underlying operating system views that file.

In the case of Unix-like operating systems, the execute permission must be set for the user running ´twander´ (or their group) for the file to be executed.

On Win32, the file will be executed if the user has permission to do so **and** that file is either executable or there is a Windows association defined for that file type. For example, double-clicking on a file ending with ".txt" will cause the file to be opened with the ´notepad´ program (unless the association for ".txt" has been changed).

If ´twander´ determines that it is running on neither a Unix-like or Win32 system, double-clicking on a file does nothing.

Run User-Defined Command
User-Defined (Single Letter) Key

Each command defined in the configuration file has a Command Key associated with it. Pressing that key will cause the associated command to be run. If no command is associated with a given keystroke, nothing will happen when it is pressed.

MENU OPTIONS

Although ´twander´ is primarily keyboard-oriented, several menu-based features are also implemented to make the program more convenient to use. These menus appear at the top of the ´twander´ display window, above the directory listing.

The first item in each menu is a dashed line ("----") which indicates that it is a "tearoff" menu. Clicking on the dashed line will detach the menu from ´twander´ allowing it to be placed anywhere on screen. Even when detatched, these menus remain current and in-sync with ´twander´ as it continues to run. You can also tear off multiple instances of these menus if you'd like copies of them at several locations on the screen simultaneously.

Commands Menu

Every command defined in the configuration file is listed in this menu by its Command Name. The association Command Key is also shown in parenthesis. Clicking on an item in this menu is the same as invoking it from the keyboard by its Command Key. This is a convenient way to invoke an infrequently used command whose Command Key you've forgotten. It is also handy to confirm which commands are defined after you've edited and reloaded the configuration file. The commands are listed in the order in which they are defined in the configuration file. This allows most frequently used commands to appear at the top of the menu by defining them first in the configuration file. If no commands are defined, either because the configuration file contains no Command Definitions or because the configuration file cannot be opened for some reason, the Commands Menu will be disabled (grayed out).

Directories Menu

´twander´ keeps track of every directory visited. The previously described command to move "Back" one directory allows directory navigation in reverse traversal order - you can back up to where you started. However, this feature "throws away" directories as it backs up, sort of like an "undo" function.

The "Directories" menu provides a slightly different approach to the same task. It keeps permanent track of every directory visited and displays that list in sorted order. This provides another way to move directly to a previously visited directory without having to manually navigate to it again, back up to it, or name it explictly using the Change Directory command.

LOCATION OF CONFIGURATION FILE

´twander´ needs a configuration file in order to define commands available to the user. Although the program will run without a configuration file present, it will warn you that it is doing so with no commands defined. Not only are commands defined in this configuration file, but keyboard bindings can optionally be assigned (changed from their defaults) in this file.

By default, the program expects to find configuration information in **$HOME/.twander** but you can override this with the -c command line option. (Recommended for Win32 systems - see the section below entitled, **INSTALLING ´twander´**)

Actually, ´twander´ can look in a number of places to find its configuration file. It does this using the following scheme (in priority order):

- If the -c argument was given on the command line, use this argument for a configuration file.
- If -c was not given on the command line, but the HOME environment variable is set, look for the a configuration file as $HOME/.twander.
- If the HOME environment variable is not set **and** a -c command line argument was not provided, look for a file called ".twander" in the directory from which ´twander´ was invoked.

CONFIGURATION FILE FORMAT

´twander´ configuration files consist of freeform lines of text. Each line is considered independently - no configuration line may cross into the next line. Whitespace is ignored within a line as are blank lines.

There are only four possible legal lines in a ´twander´ configuration file: Comments, User-Defined Variables, Key Binding Statements and Command Definitions. Everything else is considered invalid. ´twander´ will respond with errors or warnings as is appropriate anytime it encounters a problem in a configuration file. An error will cause the program to terminate, but the program continues to run after a warning.

This is both true when the program initially loads as well as during any subsequent configuration file reloads initiated from the keyboard while running ´twander´.

See the ".twander" file provided with the program distribution for examples of valid configuration statements.

Comments

A comment is begun with the "#" string which may be placed anywhere on a line. Comments may appear freely within a configuration file. ´twander´ strictly ignores everything from the "#" to the end of the line on which it appears without exception. This means that "#" cannot occur anywhere within a User-Defined Variable Definition, Key Binding Statement, or Command Definition (these are described below). Comments **can** be placed on the same line to the right of such statements.

It is conceivable that the "#" character might be needed in the Command String portion of a Command Definition. ´twander´ provides a Built-In Variable, [HASH], for exactly this purpose. See the section on Built-In Variables below for a more complete description.

User-Defined Variables

User-Defined Variables are defined using the syntax:

Variable-Name = Replacement-String

For example,

EDITOR = emacs blah blah blah blah

Later on, when defining a command, instead of typing in "emacs blah blah blah blah", you can just refer to the variable [EDITOR] - the brackets indicate you are **referring** to a previously defined variable.

Why bother with this? Because it makes maintaining complex configuration files easier. If you look in the example ".twander" configuration file provided in the program distribution, you will see this is mighty handy when setting up complex "xterm" sessions, for example.

Here are several other subtleties regarding User-Defined Variables:

- ´twander´ variable definitions are nothing more than a string substitution mechanism. Suppose you have a variable definition that refers to another variable:

  NewVar = somestring [OldVar]

  It is important to realize that this only means: "If you encounter the string ´[NewVar]´ **in a subsequent Command Definition,** replace it with the string ´somestring [OldVar]´.

  In other words, no evaluation of the right side of the expression takes place when a variable is **defined.**

Evaluation of a variable only takes place when the variable is **referenced** (in the Command String portion of a Command Definition). The Command Definition parser will continue to dereference variable names until they are all resolved or it has reached the maximum nesting level (see next bullet).

- User-Defined Variables may be **nested** up to 32 levels deep. You can have constructs like:

  Var1 = Foo
  Var2 = Bar
  FB = [Var1][Var2]

  Later on (when defining some command) when ´twander´ runs into the variable reference [FB], it will keep substituting variables until all [...] references have been resolved or it hits the nesting limit (32). This limit has to be imposed to catch silly things like this:

  Var = a[Var]

  This recursive definition is a no-no and will be cause ´twander´ to generate an error while parsing the configuration file and then terminate.

  Your variable definitions can also nest other kinds of variables (Environment and Built-Ins). So, constructs like this are perfectly OK:

  Var1 = [$PAGER]
  Var2 = command-arguments
  V = [Var1] [Var2] [DSELECTION]

- In the example above, notice that since the right-hand side of User-Defined Variables is literally replaced, we have to make sure there is space between the various variable references. If we used [Var1][Var2][DSELECTION] we would get one long string back instead of a command with arguments and a list of selected items.

- Variables must be **defined before they are referenced** (in a Command Definition). You can, however, include not-yet defined variable name in another User-Variable Definition so long as all these variable are defined by the time they appear in a Command String. The following is OK because all variables are defined by the time they are actually needed:

  Var1 = foo
  Var2 = [Var3] # This is just a string substitution, not a reference
  Var3 = bar
  MyVar = [Var1][Var2]

  # Now comes the command definition
  # If we put this before the Variable Definitions above,
  # it would be an error.

  x mycommand [MyVar]

- Variable Names are case-sensitive - [EDITOR], [Editor], and [editor] all refer to different variables.

- The "#" character cannot be used in either the variable name or the replacement string since doing so begins a comment.

- The "=" is what separates the Variable Name from the replacement string. Therefore, the "=" cannot ever be part of a Variable Name. A Variable Name cannot begin with "$" (see next bullet). Other than these minor restrictions, both Variable Names and Replacement Characters can be any string of characters of any length. Good judgment would suggest that Variable Names should be somewhat self-descriptive and of reasonable length - i.e., Much shorter than the replacement string!

- A Variable Name must never begin with "$". This is because a Command Definition containing a string in the form [$something] is understood by ´twander´ to be a reference to an **Environment Variable,** named "something". If you do this:

  $MYVAR = some-string

  You will never be able to subsequently reference it because, [$MYVAR] tells ´twander´ to look in the current environment, not its own symbol table to resolve the reference. However, note that "$" symbol may appear anywhere else but the first character of a variable name. So, for example, MY$VAR is fine.

- Variable Names may not be redefined. This means you can only define a given Variable Name once per configuration file. It is also considered a variable redefinition if you try to use a variable name which matches either one of the Built-In Variables (used in Command Definitions) or one of the Program Function Names (used for Key Bindings).

Key Binding Statements

Key Binding Statements look just like User-Defined Variables. The ´twander´ parser automatically figures out which is which. For a detailed explanation of key binding, see the section below entitled, **CHANGING KEYBOARD BINDINGS.**

Command Definitions

The heart of the ´twander´ configuration process is creating of one or more **Command Definitions.** These definitions are the way user-defined commands are added to a given instance of ´twander´. A Command Definition consists of three fields separated by whitespace:

Command-Key Command-Name Command-String

The **Command Key** is any single character which can be typed on the keyboard. This is the key that will be used to invoke the command from the keyboard. Command Keys are case-sensitive. If "m" is used as a Command Key, "M" will not invoke that command. Command Keys must be unique within a given configuration file. ´twander´ will declare an error and refuse to run if it sees two Command Definitions with the same Command Key in a given configuration file. A Command Key can never be "#" which is always understood to be the beginning of a comment.

The **Command Name** is a string of any length containing any characters. This is the name of the command which is used to invoke the command from the Command Menu. Command Names are case-sensitive ("command" and "Command" are different names), but they are not required to be unique within a given configuration file. That is, two different Command Definitions may have identical Command Names associated with them, though this is not ordinarily recommended.

The **Command String** is any arbitrary string which is what ´twander´ actually tries to execute when the command is invoked.

A Simple Command Definition

In its simplest form, a Command Definition looks like this:

```
# A simple Command Definition

m  MyMore  more somefile
```

This command can be invoked pressing the "m" key on the keyboard or selecting the "MyMore" entry from the Command Menu. Either way, ´twander´ will then execute the command, "more somefile".

The problem is that this command as written actually will not give you the result you'd like (...well, on X-Windows - is does work on Win32 as written). (For more details on why, see the **GOTCHAS** section below.) It turns out that starting a non-GUI program like ´more´ in a new window needs some extra work. What we want to do is run ´more´ inside a copy of ´xterm´. Now our command looks like this:

```
# Our command setup to run as a GUI window

m MyMore xterm -l -e more somefile
```

User-Defined Variables In A Command String

The last example works quite nicely. But, we're probably going to end up using the string "xterm -l -e" over and over again for any shell commands we'd like to see run in a new window. Why not create a User-Defined Variable for this string so we can simplify its use throughout the whole configuration file? Now, our command looks like this:

```
# Our command enhanced with a User-Defined Variable.
# Remember that the variable has to be defined *before*
# it is referenced.

XTERM = xterm -l -e              # This defines the variable
m MyMore [XTERM] more somefile   # And the command then uses it
```

Environment Variables In A Command String

This is all very nice, but we'd really like a command to be generic and be easily used by a variety of users. Not everyone likes the "more" program as a pager. In fact, on Unix-like systems there is an environment variable ($PAGER) set by each user which names the paging program that user prefers. We can refer to environment variables just like any other variable as explained previously. Now our command looks like this:

```
# Our command using both a User-Defined Variable and
# an Environment Variable to make it more general

XTERM = xterm -l -e
m MyMore [XTERM] [$PAGER] somefile
```

Built-In Variables In A Command String

It would also be really nice if the command applied to more than just a single file called "somefile". The whole point of ´twander´ is to allow you to use the GUI to select one or more directories and/or files and have your Command Definitions make use of those selections. ´twander´ uses a set of **Built-In Variables** to communicate the current directory and user selections to the any commands you've defined. Built-In Variables are referenced just like User-Defined Variables and Environment Variables and may be inserted

any appropriate place in the Command String. In our example, we probably want the command to pickup whatever item the user has selected via the GUI and examine that item with our paging program. Now our command becomes:

```
# Our command in its most generic form using
# User-Defined, Environment, and Built-In Variables

XTERM = xterm -l -e
m MyMore [XTERM]  [$PAGER]  [DSELECTION]
```

The "DSELECTION" built-in is what communicates the currently selected item from the GUI to your command when the command actually gets run.

´twander´ has a small, but rich set of Built-In Variables for use in your command definitions:

- **[DIR]**

  [DIR] is replaced with the current directory which ´twander´ is viewing.

- **[DSELECTION]**

  [DSELECTION] is replaced with the fully-qualified path name of the item currently selected in the GUI. If more than one item is selected, [DSELECTION] refers to the last item in the group (the bottom-most, not the most recent item you selected).

- **[DSELECTIONS]**

  [DSELECTIONS] is replaced with the fully-qualified path name of **all** items currently selected in the GUI.

- **[HASH]**

  Because ´twander´ always recognizes the "#" as the beginning of a comment, there is no direct way to include this character in a Command String. It is conceivable that some commands (such as ´sed´) need to make use of this character. The [HASH] built-in is provided for this purpose. Anywhere it appears in the Command String, it will be replaced with the "#" at command execution time. Unlike all the other Built-In Variables, [HASH] is never quoted when it is replaced in a Command String, regardless of whether the -t command argument is used or not.

- **[SELECTION]**

  [SELECTION] is replaced with the name of the currently selected item in the GUI. The path to that file is **not** included. As with [DSELECTION], if more than one item is selected in the GUI, the name of the last item in the group is returned for this variable.

- **[SELECTIONS]**

  [SELECTIONS] is replaced with the names of **all** items currently selected in the GUI. The path to those names is not included.

- **[PROMPT:Prompt-String]**

  [PROMPT:...] allows you to insert an interactive prompt for the user anywhere you'd like in a Command String. The user is prompted with the "Prompt String" and this variable is replaced with their response. If they respond with nothing, it is interpreted as an abort, and the command execution is terminated. This makes commands extremely powerful. For instance, say you want to create a group copy command:

  ```
  # Copy a group of items to a location set by
  # the user at runtime
  UnixCopy  = cp -R
  Win32Copy = copy

  # Unix Version
  c UnixCP [UnixCopy]  [DSELECTIONS]  [PROMPT:Enter Destination]

  # Win32 Version
  C Win32CP [Win32Copy]   [DSELECTIONS]  [PROMPT:Enter Destination]
  ```

A few other points about Built-In Variables are worth noting:

- Built-In Variables which return a directory name do **NOT** append a path separator character ("/" or "\") to the end of the name even though it is visible in the GUI. This provides maximum flexibility when defining commands. It is up to the command author to insert the appropriate path separator character where needed. (NOTE: Earlier releases of ´twander´ **did** include the trailing path separator and you may have to edit older configuration files accordingly. This change was necessary because certain commands like Unix ´cp´ will not work if given a source directory with the path separator included.)

  For example, another way to express the full path of the currently selected item is:

  ```
  # Unix Path Separator
  UPSEP = /

  #Win32 Path Separator
  WPSEP = \

  [DIR][UPSEP][SELECTION]

  or

  [DIR][WPSEP][SELECTION]
  ```

  Be aware that, because of ´twander´ quoting rules, such constructs will result in strings like:

  "/mydir"/"myfile"

  or

  "C:\mydir"\"myfile"

  This should not generally be a problem with the various Unix shells, and may work for some Win32 commands. However, some Win32 programs (noted on ´notepad´) reject this kind of file name when passed on the command line. The workaround (and a generally easier way to do this sort of thing), is

to use the [DSELECTION] built-in which returns the full path name of an item as a single quoted string.

- User-Defined and Environment Variables are processed at the time the configuration file is read by ´twander´. That is, they are handled **once** at load time.

- By contrast, Built-In Variables are resolved **on each command invocation,** i.e - at command runtime.

- The results of all built-ins (except HASH) are put inside double-quotes when they are replaced in the Command String. This default is recommended so that any built-in substitutions of, say, file names with spaces in them, will be properly recognized by your commands. You can suppress the addition of double-quotes by using the -t command line option when starting ´twander´.

- Any of the variable types may appear multiple times in the same Command String. For example, suppose you want to define a generic Unix copy command:

  g gencopy cp -R [PROMPT:Enter Source] [PROMPT:Enter Destination]

  When the user presses "g" (or clicks on "gencopy" on the Command Menu), they will be presented with two prompts, one after the other, and then the command will run.

CHANGING KEYBOARD BINDINGS

No program that runs in many operating environments can satisfy everyone's (anyone's!) idea of what the "correct" key bindings should be. An emacs user, vi user, BSD user, and Windows user are going to differ considerably on what keys should be bound to what feature.

It is not difficult to override the default keyboard bindings by adding entries in the configuration file. Doing so requires some familiarity with how Tkinter names keystrokes. Good resources for learning this exist abundantly on the Internet, among them:

http://www.pythonware.com/library/tkinter/introduction/index.htm
http://www.nmt.edu/tcc/help/pubs/lang.html
http://www.cs.mcgill.ca/~hv/classes/MS/TkinterPres/

(As an aside - Tkinter is nothing more than a Python interface to the Tcl/Tk windowing system. The "real" naming conventions for keystokes can be found in the many sources of Tk documentation, both in print and on the Internet.)

Keyboard binding assignments look just like local variable definitions in the configuration file. (The ´twander´ configuration file parser automatically distinguishes between key binding statements and user variable definitions. This means you can never use one of the program function names as one of your own variable names.) Key binding statements thus take the form:

Program-Function-Name = Tkinter-Keystroke-Name

Changing the default bindings is therefore nothing more than a matter of assigning the appropriate Program Function Name (found in parenthesis next to the description in the default descriptions above) to the desired keystroke.

Examples of all the default key bindings are shown as comments in the ".twander" example configuration file supplied in the program distribution. The easiest way to rebind a particular function is to uncomment

the relevant line and change the right side of the assignment to the new key you'd like to use. More detailed instructions on what to do are found in the example ".twander" file itself.

It is important to observe several rules when rebinding keys:

- It is best if keyboard navigation commands are all Control or Function keys. If you assign a navigation or selection function to a single keystroke, it may conflict with a user-defined command. If you assign it to a keypad/special key it may conflict with that key's normal GUI behavior.

- The Tkinter keynames should placed on the right side of the "=" symbol **without any quotation marks.**

  So, this is correct: QUITPROG = <F3>

  But, this is not: QUITPROG = '<F3>'

- The Program Function Name variables (the left side of the assignment) may not be used as names for your own user-defined variables elsewhere in the configuration file. In fact, ´twander´ will never even recognize such an attempt. For example, suppose you try to do this:

  QUITPROG = something-or-other

  Because you want to be able to reference [QUITPROG] in a subsequent command definition. ´twander´ will actually interpret this as just another key binding command, in this case binding the program function QUITPROG to "something-or-other" - probably not what you intended. Moreover, if you have a Command String somewhere with [QUITPROG] in it, ´twander´ will declare and error and abort because it has no User-Defined variable of that name in its symbol table.

- When you're done making changes to the configuration file, be sure to either restart the program or reload the configuration file to assign the new bindings.

- Be aware that ´twander´ does no sanity testing on the assignments you change. If you assign a particular ´twander´ function to an illegal or silly key string, the program will probably blow-up spectacularly. At the very least, that program feature will probably be unusable, even if ´twander´ manages to run.

GOTCHAS

There are several tricky corners of ´twander´ which need further explanation:

1) Getting Command Results Displayed In A New Window

When you invoke a command via ´twander´, whether via a command definition in the configuration file or the keyboard shortcut, you generally want it to run in a new window. If the program you are running is GUI-aware, this should not be a problem. However, if you are using ´twander´ to run a command line program or script, you have to take extra care in the formulation of the Command String. In the case of Unix-like systems you have to invoke the command so that it runs in some GUI context. Say you want to use a pager like ´less´ to view files. You would expect that this entry might do it:

V view less [DSELECTIONS]

Sadly, this will not work, at least not the way you expect. If you started ´twander´ from a terminal session and use the command above, it will work, but the results will appear in the invoking terminal window, **not**

in a new window as you might expect. If you started ´twander´ from a GUI or disconnected it from the initiating terminal with a ´nohup´ ... & invocation, you will get **no** output. This is not a ´twander´ problem, it is innate to how command line programs run under Unix-like shell control. To achieve the desired results, you'll need something like this in your configuration file:

V view xterm -l -e less [DSELECTIONS]

This causes your command line program to execute in an ´xterm´ context.

This is not so much of an issue on Win32 systems where the first form of the command above works fine.

2) Modal Operation Of New Windows

Notice our example commands above do not end with "&". These should not be needed on either Unix-like or Win32 operating systems. When a command is executed, ´twander´ starts a new thread of execution which runs concurrently with ´twander´ itself. This means you should be able to continue using ´twander´ while the new command executes. If not (´twander´ is locked out while the new command runs - so-called "modal" operation), it means your system does not completely or correctly implement threading. In that case, if you want non-modal command operation, try adding a "&" at the end of your command definition.

3) Windows That Don't Disappear On Command Completion

It appears that some X Windows implementations (noted on XFree86 / FreeBSD) do not correctly destroy an ´xterm´ window after a command initiated with -e terminates. This is not a ´twander´ problem. The workaround is to manually kill the window or press enter once when the command is complete and the window has input focus.

OTHER

You must have Python 2.2 or later installed as well as Tkinter support installed for that release. In the case of Win32, Tkinter is bundled with the standard Windows Python distribution. In the case of Unix-like systems, you may have to first install Python and then the appropriate release of Tkinter. This is the case, for example, with FreeBSD.

BUGS AND MISFEATURES

The color options (-b, -f), font options (-n, -s, -w), and size option (-x, -y) are **not** checked for validity when the command line is initially read. If you enter something unreasonable for these options, ´twander´ will refuse to run with some **really** interesting and entertaining error messages. The program could be more gracious about this.

The configuration file parser does no validation of key binding override values. It is entirely possible to bind a ´twander´ feature to a bogus key definition. This will cause either a spectacular program failure or, at the very least, that feature will not work correctly or at all. The assumption here is that if you are smart enough to want to change key bindings, you're smart enough to learn how Tkinter names keys. You have been warned.

This program has not been tested on MacOS. Please let me know how/if it works there and any issues you discover.

INSTALLING ´twander´

Installation of ´twander´ is fairly simple and takes only a few moments. The most important thing before installing the program is to make sure you have Python 2.2 (or later) with Tkinter support installed on your system.

One other note: However you install the program, it is probably easiest to get started by editing the example ".twander" file to taste. Be aware that this file is shipped with everything commented out. You have to uncomment/edit the section relevant to your operating system: Unix-like or Win32.

Installing Using The FreeBSD Port

If you've installed ´twander´ using the FreeBSD port, all you have to do is copy the example configuration file, ".twander" found in /usr/local/share/doc/twander to your home directory and edit it to taste.

Make sure that /usr/local/bin is in your path. To start the program, just type "twander.py" from the shell prompt.

Installing Manually On A Unix-like System

Copy the "twander.py" file to a directory somewhere on your path. (/usr/local/bin is a good candidate). Make sure this file has permissions 755 and owner/group appropriate for your system (root/wheel, root/root, or bin/bin). Copy the ".twander" file to your home directory and edit to taste.

To run the program, just type "twander.py" from a shell prompt.

Installing Manually On A Win32 System

Copy the "twander.py" file to a directory somewhere on your path, or create a new directory to hold this file and add that directory path to the PATH environment variable.

IMPORTANT NOTE TO WIN32 USERS: Windows has the old MS-DOS legacy of assuming that a "." begins a file "extension". Although you can create and read files in the form ".something", it is not recommended because many Win32 programs get confused when they see this. It is also difficult to remove files named this way with the standard Windows programs and utilities. This is especially the case for older Win32 operating systems like Win98. For this reason, it is recommended that you rename the ".twander" default configuration file provided in the program distribution to something else like "twander.conf" and use the ´twander´ -c command line option to point to this configuration file.

On Win32, where to put the configuration file raises an interesting question. Microsoft operating systems normally do not set the "HOME" environment variable, because they have no notion of a "home" directory - Well, they do, but it is called "USERPROFILE" not "HOME". So, you can either create a new user-specific environment variable called HOME yourself (which points to your desired home directory) or you can invoke ´twander´ with the -c argument to explictly declare where it can find its configuration file.

You can run the program several ways on Win32 systems:

- Create a Win32 shortcut which points to the "twander.py" file using the "pythonw" command to invoke it. Normally, starting a Python program from the Windows GUI creates a parent window which persists as long as the program runs. Using "pythonw" instead of "python" to run your program suppresses the creation of this blank parent window. For example, you might have something like this in the "Target:" field of your shortcut:

  "C:\Program Files\Python22\pythonw.exe" C:\twander.py /

  This runs the program starting at the root directory of the current drive (assuming "twander.py" is located in C:\".

- Start a command line window and issue a command like the one above directly from the command line.

- Use Windows Explorer (or better still, an already running instance of ´twander´!) to navigate to the directory where "twander.py" is located. Double-click on the file. If Python is properly installed, there should be an association for ".py" file types and ´twander´ should start automatically.

DESIGN PHILOSOPHY

Graphical User Interfaces (GUIs) are a blessing and a curse. On the one hand, they make it easy to learn and use a computer system. On the other, they are a real inconvenience to experienced users who are touch typists. Taking hands off the keyboard to use the mouse can really slow down a good typist.

Nowhere is this more apparent than in filesystem browsers. In one corner we have the GUI variants like ´Konqueror´ and ´Microsoft Windows Explorer´. These are very easy to use but you pretty much need the mouse in your hand to do anything useful. In the other corner are the text-based file browsers like ´List´, ´Norton Commander´, and ´Midnight Commander´. These are really efficient to use, but have limited functionality and generally do not operate very well on **groups** of things.

Both of these approaches also suffer from the well-known interface problem of "What You See Is **All** You Get" - Each program has a predefined set of commands and the user cannot easily extend these with their own, new commands.

´twander´ is another approach to the filesystem navigation problem which embraces the best of both the GUI-based approach and the text-based approach. It also provides a rich mechanism whereby each user can easily define their own command set and thereby customize the program as they see fit. This is done with a number of key features:

1) The **Navigation** of the filesystem is graphical - you can use the mouse to select files, directories, or to change directories. However, each major filesystem navigational feature is also doubled on the keyboard (using Control keys) so you can move around and select things without ever touching the mouse.

2) ´twander´ also supports a number of **navigation shortcuts.** It provides single control-key access to changing directories, moving to the previous directory, moving up one directory level, moving to any previously visited directory, (de)selecting any or all files/directories in the current view, and escaping to the operating system to run a command. Some (but not all) of these features are also doubled via GUI/mouse operations.

3) There are **no** built-in file or directory commands. All commands which manipulate the files or directories selected during navigation are user-defined. This command definition is done in an external configuration file using a simple but powerful command macro language. This means that that the command set of the program can easily be changed or expanded without having to release a new version of ´twander´ every time. Better still, every different user can have their own command set defined in a way that suits their style of working. Best of all, commands can be invoked either graphically (with a mouse click) or via a single keypress to minimize moving your hands off the keyboard.

4) Because ´twander´ is written in Python using Tkinter, the same program runs essentially identically on many Unix-like and Win32 systems. The only thing that may need to be changed across these various platforms are the command definitions in the configuration file. You only need to learn one interface (and the commands you've defined) across all the different systems you use.

The consequence of all this is that ´twander´ is an extremely powerful and highly customizable filesystem navigator. Once learned, both navigation and command execution are lightning-fast (or at least, as fast as

your machine can go ;) while minimizing dependency on the mouse.

COPYRIGHT AND LICENSING

AUTHOR

Tim Daneliuk
twander@tundraware.com