_goto.py

someonesdad1@gmail.com 4 Feb 2019

I use this python script with some shell functions to open remembered project files and to navigate to remembered locations on my computer. The key advantages are:

- ♦ It can be used for multiple projects, each with its own datafile for remembered paths.
- ♦ The remembered paths can be commented out so you don't see them (and, thus, they don't clutter up the working display). The benefit is that you can look at the datafile later to find where a particular project was stored.

Here's an example of the basic use. I use a shell function named `pr` to keep track of the projects I'm working on. When I type `pr` at the command line, I might see a listing like

```
1   /pylib/pgm/_goto.odt
2   Project2

di        Ditch project
Selection?
```

The script is prompting me to choose which application (i.e., remembered file name) I wish to launch. Just pressing the return key takes me back to a shell prompt. The numbered directory names (or their aliases) are those that don't have a string shortcut given to them. The lower list of directories are those that have a string shortcut.

If I remember the associated number or string shortcut, I can type that in as an argument to the `pr` command and go to that directory directly. I use this feature a lot because the last-saved directory is the first in the numbered list.

If I type in `pr e`, the project's datafile is called up in my editor. This lets me reorder things, delete things, comment out an entry (saving it but not displaying it), give a file path an alias, or give an entry a string shortcut.

Here's a datafile that could result in the above display:

```
/pylib/pgm/_goto.odt
Project2; /pylib/project2.odt
Ditch project;di;/doc/ditch.odt
Hidden project;@hi;/doc/hidden.odt
#Old ditch project;di;/doc/ditch_old.odt
```

The first line only contains a file name. If you enter the number 1 at the `Selection?` prompt, the `_goto.py` script will cause this file to be opened with its associated application.

The second line uses an alias for the file to open.

The third line uses both an alias and a string shortcut. I thus can open the ditch project's file by typing `pr di` at the command line.

The fourth line uses the `@` character to flag a project whose file is not to be displayed. I use this for projects that I open a lot; I choose not to see them listed because I remember their shortcut string.

The last line is commented out because of the `#` character. This lets me "remember" the location of a project file, but not see it displayed if I'm not actively working on it. A benefit of this is that it lets me find an old project file. My computer has over 60,000 directories on it and it's not hard to forget where a particular file is.

When I want to remember a project file, I use the command `pr a <path_to_file>`. I'll often follow this up with a `pr e` command to edit the `.projectrc` file to give the directory an alias or a string shortcut. A convenience is that the most-recently-remembered project file is added to the beginning of the file, making it number 1 on the printed list.

Two handy options to the `pr` command are `-t` and `-T`. These commands have the `_goto.py`

script check that each active or remembered path, respectively, is a valid path in the file system. This is useful over long periods of time because you'll often rename or move things and then not be able to find something later. If you run these options periodically, you'll identify paths that are no longer valid so you can fix them.

Shell functions

I work in a UNIX-like environment such as cygwin under Windows or Linux. I use the bash shell by default; you might have to modify these functions if you use a different shell.

Projects

I use the following shell function for my `pr` command:

```
pr()
{
     applaunch project "$@"
}
```

The reason for this is that I have multiple commands like this that launch from various project files. For example, `vid` is used to keep track of video projects.

The work is done in the `applaunch` function:

```
#-------------------------------------------------------------------------------
# Function to launch applications:
#   applaunch application     Launches application
#   applaunch a appl          Adds application appl to storage
#   applaunch e               Edits the storage file
#   applaunch S string        Search file for a string
#   applaunch -t              Check the paths in the file
#   applaunch -T              Check all paths, even those commented out
#   applaunch v appl          Use vi to edit the indicated appl
#   applaunch                 Requests an integer to launch a listed appl
#
# Parameters:
#   1   name (used to make tmp file and get rc file)

applaunch()
{
    typeset tmp=/tmp/${1?Need name of rc file}.$$
    typeset dir
    typeset appfile="$HOME/.${1}rc"
    typeset logfile="$HOME/.${1}rc.log"
    #typeset app="/usr/bin/exo-open"        # Ubuntu
    #typeset app="/home/Don/bin/app.exe"    # Windows XP days
    typeset app="cygstart"                  # Under cygwin
    typeset edit=""
    typeset goto=_goto.py
    typeset PYTHON="$PYTHON3"
    typeset PYTHONPATH="$PYTHONPATHcyg"
    typeset PYTHONLIB="$PYTHONLIBcyg"
    typeset PYTHONPGM="$PYTHONPGMcyg"
    typeset PYTHONSTARTUP="$PYTHONSTARTUPcyg"
    shift
    if [ $# -eq 0 ] ; then
        # If no arguments were given, use the rc file's entries &
        # prompt the user.
        appl="$($PYTHON $PYTHONPGM/$goto -c $appfile 0)"
        [ "$appl" = "" ] && return
    else
        if [ "$1" = "-t" ] ; then
            # Check the paths in the file
            $PYTHON $PYTHONPGM/$goto -t $appfile
            return
        fi
        if [ "$1" = "-T" ] ; then
```

```
            # Check all the paths in the file
            $PYTHON $PYTHONPGM/$goto -T $appfile
            return
        fi
        case "$1" in
            a|add) # Add application to beginning of file
                echo $($PYTHON $PYTHONPGM/abspath.py \
                       ${2?Need a file name}) >$tmp
                cat $appfile >>$tmp
                cp $appfile $HOME/.bup/${1}.$$  # Make a backup copy
                mv $tmp $appfile
                return;;
            -h|h|help) # Show the commands
                cat <<EOF
Commands:
    a filepath      Adds application filepath to config
    e               Edits the config file
    S string        Search config file for a string
    T string        Search config file for a string but only active items
    -t              Check the paths in the config file
    -T              Check all paths in the config file (even commented-out)
EOF
                return;;
            e|ed|edit) # Edit the storage file
                $EDITOR $appfile
                return;;
            T) # Search the config file for a string (active items only)
                sed -e 's/^ *//' <$appfile | \
                  grep -v ' *#' | grep -i $2
                return;;
            S) # Search the config file for a string
                sed -e 's/^ *//' <$appfile | grep -i $2
                return;;
            *)  typeset path="$($PYTHON $PYTHONPGM/abspath.py $1)"
                if [ -r "$path" ] ; then
                    echo "$(dt) $path" >>$logfile
                    $app "$path"
                else
                    appl="$($PYTHON $PYTHONPGM/$goto -c $appfile $1)"
                    echo "$appl $path" >>$logfile
                    [ "$appl" = "" ] && return
                    $app "$appl"
                fi
                return;;
        esac
    fi
    [ "$appl" = "" ] && return
    $app "$appl"
}
```

The underlying work is done by the _goto.py script, which was originally written to help me navigate/remember the various directories I worked in.

Directories

The _goto.py script was written in the late 1990's to help me remember directories on the computers I worked on. It was more powerful than the tools built into the various shells and many of the folks I worked with started using it once I showed it to them.

I use the shell function g for this task:

```
#-------------------------------------------------------------------------
# Function to allow moving around to saved directories.  Arguments are:
#   a       Adds current directory to list
#   e       Edits list
#   g n     Goes to the nth directory.  If n is not a number, it is a string
#           that is a shorthand description and is separated from the optional
#           description by a ! character.
```

```
#   -t      Checks each directory in the file
#   -T      Checks all directory in the file, even those commented out
#   -s      Print silent link names
#   S       Search all lines in the config file for a string
#   s       Search the active lines in the config file for a string

g()
{
    # Note this uses cygwin's python 3
    typeset dir
    typeset GOTO=_goto.py
    typeset GOTORC="$HOME/.gotorc"
    # Above works in Linux; following is for Windows
    typeset GOTORC="c:/cygwin/home/donp/.gotorc"

    if [ $# -eq 0 ] ; then
        # Pass the script the 0 argument to have it prompt the user
        PYTHONLIB=$PYTHONLIBcyg PYTHONPATH=$PYTHONPATHcyg \
           dir="$($PYTHON3 $PYTHONPGMcyg/$GOTO $GOTORC 0)"
    else
        if [ "$1" = "home" ] ; then
            cd $(cat $HOME/.curdir)
            return
        fi
        case $1 in
            a)  # Add the current directory to the head of the goto file
                typeset tmp=/tmp/g.$$
                echo $(pwd) >$tmp
                cat $GOTORC >>$tmp
                cp $GOTORC $HOME/.bup/goto.$$  # Make a backup copy
                mv $tmp $GOTORC
                if [ $? -ne 0 ] ; then
                    echo "Addition failed"
                    # Restore from the backup copy
                    cp $HOME/.bup/goto.$$ $GOTORC
                fi
                return
                ;;
            e)  # Edit the goto file
                $EDITOR $GOTORC
                return
                ;;
            -h) # Print a help message
                cat <<EOF
a       Add the current directory to the head of the goto file
e       Edit the goto file
g n     Go to the nth directory in the listing
S       Search active lines for a regular expression
S       Search all lines for a regular expression
-t      Check the goto file's directories exist
-T      Check all the goto file's directories exist, even those commented-out
-s      Print silent link names

Otherwise, the item on the command line is looked up.
EOF
                return
                ;;
            S)  # Search the whole config file for a string
                grep -i $2 $GOTORC
                return;;
            s)  # Search active lines for a string
                grep -v "^[ \t]*#" $GOTORC | grep -i $2
                return;;
            -t) # Check the directories
                PYTHONLIB=$PYTHONLIBcyg PYTHONPATH=$PYTHONPATHcyg \
                   $PYTHON3 $PYTHONPGMcyg/$GOTO -t $GOTORC
                [ $? -eq 0 ] && echo "$GOTORC directories OK"
                return
```

```
                ;;
            -T) # Check all the directories
                PYTHONLIB=$PYTHONLIBcyg PYTHONPATH=$PYTHONPATHcyg \
                    $PYTHON3 $PYTHONPGMcyg/$GOTO -T $GOTORC
                [ $? -eq 0 ] && echo "$GOTORC all directories OK"
                return
                ;;
            -s) # Print silent link names
                PYTHONLIB=$PYTHONLIBcyg PYTHONPATH=$PYTHONPATHcyg \
                    $PYTHON3 $PYTHONPGMcyg/$GOTO -s $GOTORC
                return
                ;;
            *)  # cd to the choice given on the command line
                PYTHONLIB=$PYTHONLIBcyg PYTHONPATH=$PYTHONPATHcyg \
                    dir="$($PYTHON3 $PYTHONPGMcyg/$GOTO $GOTORC $1)"
                ;;
        esac
    fi
    # The following removes any carriage returns, which can happen under
    # Windows.
    dir="$(echo $dir | tr -d '\r')"
    # cd to the desired directory; if it has a .profile file and is
    # not our home directory, source it (this is useful for project
    # aliases, etc.).
    if [ "$dir" ] ; then
        cd "$dir"  # Note 'cd -' works as expected
        typeset p=.profile
        if [ -r $p -a "$(pwd)" != "$HOME" ] ; then
            # First, make sure it's our file
            owner=$(ls -l $p | awk '{print $3}')
            if [ "$owner" = "$LOGNAME" ] ; then
                . $p
                echo "Sourced $p"
            fi
        fi
    fi
}
```

You may have to fiddle with the commands in this script to get things to work on your system. In particular, I use various versions of python, so environment variables like `PYTHONLIBcyg`, `PYTHONPATHcyg`, and `PYTHON3` keep track of the details.