Dotfile madness

We are no longer in control of our home directories.

My own home directory contains 25 ordinary files and 144 hidden files. The dotfiles contain data that doesn't belong to me: it belongs to the programmers whose programs decided to hijack the primary location designed as a storage for my personal files. I can't place those dotfiles anywhere else and they will appear again if I try to delete them. All I can do is sit here knowing that in the darkness, behind the scenes, they are there. Waiting in silence. Some of those programmers decided to additionally place some normal files and directories in the same place. Those are clearly visible every time I execute `ls` in my home directory. It is beyond me why my home directory ended up up containing a `node_modules` directory, `package-lock.json`, a `yarn.lock` file (I have never even consciously used yarn!), some 2 strange log files origination from some Java software clearly using an H2 database, and a `Desktop` directory. That last one has been created by Steam, which is quite unfortunate as I simply do not have a desktop or a desktop environment on my machine. I dread the day in which I will hear a loud knock on my door and one of those programmers will barge in informing me that he is going to store a piece of his furniture in the middle of my living room, if I don't mind.

To those of you reading this: I beg you. Avoid creating files or directories of any kind in your user's `$HOME` directory in order to store your configuration or data. This practice is bizarre at best and it is time to end it. I am sorry to say that many (if not most) programs are guilty of doing this while there are significantly better places that can be used for storing per-user program data.

Even if we will never be able to solve this problem - due to the historical baggage, backward compatibility, software no longer receiving updates or programming villains storing the files wherever they want just to see the world burn - we can at least try to follow some sane practices. While the mistake of introducing the mere concept of a "hidden" file can't be undone anymore, we can at least try to mitigate its results.

This particular problem has been noticed and solved a long time ago with the creation of XDG Base Directory Specification. The specification defines a set of environment variables pointing programs to a directory in which their data or configuration should be stored. It is up to the user to set those variables so if the variables are not available the programs are expected to default to a directory defined by the standard and not the user's home directory.

User-level variables

`$XDG_DATA_HOME`

> `$XDG_DATA_HOME` defines the base directory relative to which user specific data files should be stored. If `$XDG_DATA_HOME` is either not set or empty, a default equal to `$HOME/.local/share` should be used.

Example usage: storing plugins downloaded by the user, databases created by your program, user's input history, bookmarks, emails, and so on.

`$XDG_CONFIG_HOME`

> `$XDG_CONFIG_HOME` defines the base directory relative to which user specific configuration files should be stored. If `$XDG_CONFIG_HOME` is either not set or empty, a default equal to `$HOME/.config` should be used.

You should use this directory to store user-specific configuration files for your program. You will most likely want to create a default configuration file with sane and sensible default values the first time your program is executed.

`$XDG_CACHE_HOME`

> `$XDG_CACHE_HOME` defines the base directory relative to which user specific non-essential data files should be stored. If `$XDG_CACHE_HOME` is either not set or empty, a default equal to $HOME/.cache should be used.

Example: caching thumbnails generated by your file manager, songs that the user of a music straming software is often listening to, and so on. Your program should continue to function without any problems if this directory is removed by the user. Ensure that the files that are no longer needed are properly removed. Remember that exceeding the sensible amount of storage space used by your files will most likely upset the user who will quickly track down your program as a culprit.

`$XDG_RUNTIME_DIR`

> `$XDG_RUNTIME_DIR` defines the base directory relative to which user-specific non-essential runtime files and other file objects (such as sockets, named pipes, ...) should be stored.

The specification lists a series of requirements that have to be fulfilled by this directory. As described by the specification you should use this directory to store sockets and similar files used for communication.

System-level variables

`$XDG_CONFIG_DIRS`

> `$XDG_CONFIG_DIRS` defines the preference-ordered set of base directories to search for configuration files in addition to the `$XDG_CONFIG_HOME` base directory. The directories in `$XDG_CONFIG_DIRS` should be seperated with a colon ':'.
>
> If `$XDG_CONFIG_DIRS` is either not set or empty, a value equal to `/etc/xdg` should be used.

This directory should be used for storing system-wide configuration files that can be overwriten by user-specific configuration files. This directory would most likely be populated during the installation process.

`$XDG_DATA_DIRS`

> `$XDG_DATA_DIRS` defines the preference-ordered set of base directories to search for data files in addition to the `$XDG_DATA_HOME` base directory. The directories in `$XDG_DATA_DIRS` should be seperated with a colon ':'.
>
> If `$XDG_DATA_DIRS` is either not set or empty, a value equal to `/usr/local/share/:/usr/share/` should be used.

Example: storing plugins or themes that can be used by all users of your program. This directory would most likely be populated during the installation process.

How does this work in practice?

Using the standard is very simple. Read the relevant environment variable and use the default paths defined by the standard if it is missing. You should then append a program-specific directory name to it and create the entire directory tree to store your data.

As an example if you were to store configuration files you should use `$XDG_CONFIG_HOME/your-program` as your base configuraion directory instead of just storing your files directly in `$XDG_CONFIG_HOME`. Remember to never hardcode the directories to the default values defined by the standard. Read the environment variable first to allow the user to move those directories if needed.

You can easily migrate your existing programs to use this standard. In order to do so start using the standard when creating new files but keep also checking the old location of the files when reading them. This will allow you to migrate without breaking the program for the users with the configuration or data files that were created by a previous version of your program.

Read the standard to find out more and have a look at the directory hierarchy which is almost certain to be already present in your home directory. In reality a cross-platform library allowing you to get a directory for storing your data will be available for your programming language of choice. On Linux and similar systems, this library will surely use the XDG Base Directory Specification.

2019-02-01