WIP = Work in Progress

Efficient CLI Navigation

The CLI (Command Line Interface) is a very powerful, flexible and programmable environment. If you use the command line interface on a regular basis, you know how important it is to customise your working environment and develop shortcuts to ensure efficient workflow. Below are some tips on navigating through directories in an quick and easy way.

Go Back Home

The `cd` command changes the current working directory to your `/home`:

```
$ pwd
/home/user/data/documents/work
$ cd
$ pwd
/home/user
```

Go Back to the Previous Directory

To go back to the previous directory, you can use `cd -`:

```
$ pwd
/home/user/.i3/config
$ cd ~/data/projects/dotfiles/i3
$ pwd
/home/user/data/projects/dotfiles/i3
$ cd -
/home/user/.i3/config
$ pwd
/home/user/.i3/config
```

Use the Last Argument of the Previous Command

The `$_` variable returns the last argument of the previous command. This can be helpful in a variety of scenarios:

```
$ pwd
/home/user/downloads/
$ cp i3status.tar.gz ~/data/builds/i3/i3status
$ cd $_
$ pwd
```

```
/home/user/data/builds/i3/i3status
```

```
$ chmod +x /path/to/my/script/script.sh
$ $_
(This will execute script.sh)
```

The same can be achieved using the combination Alt+. or Esc+..

Bash Completion

Bash offers TAB completion, a very useful feature that reduces the number of keystrokes you are required to type to navigate to a directory.

```
$ cd d[TAB]
```

It will expand the path with a directory starting with `d`. Please note that Linux is case-sensitive so `~/Desktop` or `~/Downloads` will be ignored. If there are more than 1 directory starting with `d`, you need to press the TAB twice to get the possible completions listed below:

```
$ cd d[TAB][TAB]
data/    downloads/
```

Now press o and TAB to expand the path to `downloads`.

It can greatly reduce the number of keystrokes when accessing directories. For example:

```
$ cd data/projects/python-dir/euler/completed/
```

The keystrokes I used:

```
cd da[TAB]p[TAB]p[TAB]e[TAB]c[TAB]
```

If each directory contained only one subdirectory, I could have done the following:

```
cd [TAB][TAB][TAB][TAB][TAB]
```

Not only does the bash completion reduce the number of key strokes, but it also comes handy if you do not remember directory names.

Please note that bash completion works also on files:

```
$ cp data/projects/scripts/sync-script.sh .
```

Keystrokes used:

```
$ cp d[TAB]p[TAB]s[TAB]s[TAB] .
```

The dot (.) represents the current directory so the command will copy `sync-script.sh` to the

current directory.

Programmable Bash Completion

To take advantage of all Bash completion features, you need to install an additional package from `/extra`:

```
# slackpkg install bash-completion
```

Fix Me!

CDPATH

If you work in certain directories on a regular basis, you might want to include them in the $CDPATH variable. Suppose you often work in the `slackbuilds` directory which contains some builds:

```
$ cd ~/data/projects/slackbuilds/
$ ls
i3 i3status yajl dmenu libev
```

Add it to the $CDPATH variable by modifying `~/.bashrc`:

```
CDPATH=$CDPATH:~/data/projects/slackbuilds/
```

Please note the path included in the CDPATH variable is the path to the **parent directory** where the said directories are located.

After you have sourced `.bashrc` (`source ~/.bashrc`), you can `cd` to any of those directories from any place:

```
$ pwd
/home/user/.config/xfce4
$ cd yajl
/home/user/data/projects/slackbuilds/yajl
```

If you would like to enable Tab completion within the directories added through the CDPATH variable, you need to install `bash-completion` from Slackware's `/extra` directory.

It is NOT recommended to give your directories names similar to the system ones, eg. `usr`, `etc`, which can result in unpredictable behaviour.

Symlinks

In some situations you may consider using symlinks to create *shortcuts* to regularly visited directories:

```
$ ln -s /home/user/data/projects/scripts/slackbuilds ./slackbuilds
```

Bash Aliases

You can make your life easier by creating aliases (= shortcuts) for commands that you use often. The syntax is very simple:

```
name_of_the_alias='value'
```

You can place your aliases in `~/.bashrc`. You might need to create this file. Each time you edit this file you need to `source` it afterwards for the changes to take effect:

```
source ~/.bashrc
```

or

```
. ~/.bashrc
```

When it comes to navigation, one could, for example, create a few aliases to speed up navigating up the directory tree:

```
alias 1.='cd .. ; pwd'
alias 2.='cd ../.. ; pwd'
alias 3.='cd ../../.. ; pwd'
alias 4.='cd ../../../.. ; pwd'
```

The value of an alias can be quite complex. As you can see, `4.` will first change directories `cd ../../../..` and then print the current working directory - `pwd`. Please note a semi-colon (`;`) separating the commands.

```
$ cd data/projects/python-dir/euler/
$ 4.
/home/user
$ cd -
/home/user/data/projects/python-dir/euler
$ 3.
/home/user/data
```

Aliases can be used in a number of different ways. A few more examples:

```
alias epyt='emacs -nw /home/user/data/projects/python-dir/euler/32-
problem.py'
alias slacktop='ssh user@slacktop'
```

Directory Stack

BASH features some helpful directory stack buildins that help you navigate recently visited directories.

- `pushd` - push a directory into the directory stack and `cd` to it.
- `popd` - remove a directory from the directory stack and `cd` to it.
- `dirs` - display the list of the directories in the stack.

How does it work in practice?

First of all, add a directory to the stack. Please note that it also automatically switches to the directory (the `-n` flag suppresses this behaviour).

```
user@darkstar:~$ pushd data/projects/programming/
~/data/projects/programming ~
user@darkstar:~/data/projects/programming$
```

Alternatively, you can `cd` to a given directory and issue:

```
pushd .
```

After adding a few directories you can display the content of the stack:

```
user@darkstar:~$ dirs -v
0 ~/projects/web-develop/project-eden/includes
1 ~/projects/web-develop/project-eden/includes
2 ~/projects/web-develop/project-eden/pages/en
3 ~/projects/web-develop/project-eden/css
4 ~/projects/designs
5 ~/projects/notes
6 ~/public_html/project_eden
```

Please note that the first entry always displays the current working directory so if it also sits at the top of the stack, you'll see what seems like duplicate lines. The `-v` flag is responsible for a nicely indexed output.

To switch to one of the directories in the stack you could issue:

```
user@darkstar:~$ cd $(dirs +2 -l)
user@darkstar:~/projects/web-develop/project-eden/pages/en$
```

Admittedly, this is not the most concise way of changing directories. To make it shorter we can add an alias and a function to the `~/.bashrc` file.

```
alias dv='dirs -v'
```

List the current stack by simply typing `dv`.

```
cdd()
{
    position=$1
    if [ -z $position ]; then
        echo "You need to specify the position of the directory in the
stack"
    else
        cd $(dirs +$1 -l)
    fi
}
```

The cdd function[1] makes it possible to cd to a given directory from the stack by typing:

```
cdd 3
```

Sources

- Original source: Blog Post written by sycamorex

howtos, cli, cd, bashrc, completion, work in progress, author sycamorex

1)
Based on this function.