zplug's tags

Tag	Description	Value (default)	Example
`as`	Specify whether to register as commands or to register as plugins	`plugin`,`command`(`plugin`)	`as:command`
`of`	Specify the pattern to source files (for `plugin`) or specify relative path to add to the `$PATH` (for `command`) / In case of `from:gh-r`, can specify `of:"*darwin*{amd,386}*"` and so on	*glob* (`of:"*.zsh"`)	`of:bin`,`of:"*.sh"`,`of:*darwin*`
`from`	Specify the services you use to install	`github`,`bitbucket`, `gh-r`,`gist`, `oh-my-zsh`,`local`(`github`)	`from:gh-r`
`at`	Support branch/tag installation	*branch/tag*(`master`)	`at:v1.5.6`
`file`	Specify filename you want to rename (only `as:plugin`)	*filename* (-)	`file:fzf`
`dir`	Installation directory which is managed by zplug	**READ ONLY**	`dir:/path/to/user/repo`
`if`	Specify the conditions under which to run `source` or add to `$PATH`	*boolean* (-)	`if:"[ -d ~/.zsh ]"`
`do`	Run commands after installation/update	*commands* (-)	`do:make install`
`frozen`	Do not update unless explicitly specified	0,1 (0)	`frozen:1`
`commit`	Support commit installation (regardless of whether the`$ZPLUG_SHALLOW` is true or not)	*revision* (-)	`commit:4428d48`
`on`	Dependencies	**READ ONLY**	`on:user/repo`
`nice`	Priority of loading the plugins. If this tag is specified 10 or more, zplug will load plugins after `compinit` (see also#26)	-20..19 (0)	`nice:19`
`ignore`	Similar to `of` tag, specify exception pattern so as not to load the files you want to ignore (see also #56)	*glob* (-)	`ignore:"some_*.zsh"`

zplug configurations

ZPLUG_HOME

Defaults to `~/.zplug`. `zplug` will store/load plugins in this directory. The directory structure is below.

```
$ZPLUG_HOME
|-- bin
|   `-- some_command -> ../repos/username_A/reponame1/some_command
`-- repos
    |-- username_A
    |   |-- reponame1
    |   |   |-- README.md
    |   |   `-- some_command
    |   `-- reponame2
    |       |-- README.md
    |       `-- some_plugin.zsh
    `-- username_B
        `-- reponame1
```

If you specify `as:command` in `zplug` command, zplug will recognize the plugin as a command and create a symbolic link of the same name (if you want to rename it, set `file:` tag) within `$ZPLUG_HOME/bin`. Because zplug adds `$ZPLUG_HOME/bin` to the `$PATH`, you can run that command from any directories.

ZPLUG_THREADS

The number of threads zplug should use. The default value is 16.

ZPLUG_PROTOCOL

Defaults to HTTPS. Valid options for `$ZPLUG_PROTOCOL` are HTTPS or SSH. Unless you have a specific reason, you should use the HTTPS protocol.

For more information, see also **Which remote URL should I use?** - GitHub Help

ZPLUG_SHALLOW

Defaults to `true`. When cloning a Git repository, there is an option to limit the amount of history your clone will have. If you set this environment variable to `true`, you get the least amount of history, and you create a shallow clone.

ZPLUG_FILTER

Defaults to `fzf-tmux:fzf:peco:percol:zaw`. When `--select` option is specified, colon-separated first element that exists in the `$PATH` will be used by zplug as the interactive filter. The `ZPLUG_FILTER` also accepts the following values: `fzf-tmux -d "10%":/path/to/peco:my peco`.

ZPLUG_EXTERNAL

Defaults to `$ZPLUG_HOME/init.zsh`. This file is used to add plugins from zplug on the command-line.

ZPLUG_USE_CACHE

Defaults to `true`. If this variable is set, zplug comes to use a cache to speed up when it will load plugins after the first. The cache file is located in `$ZPLUG_HOME/.cache`. If you want to clear the cache, please run `zplug clear` or do the following:

```
$ ZPLUG_USE_CACHE=false zplug load
```

```
################################################################################
#                                                                              #
# Filename: $dotDir/bin/nmdf    (a bootstraper to install dotfiles.)           #
# Maintainer: march <titepweb@gmail.com>                                       #
#        URL: https://github.com/titepweb/nmdf                                 #
#                                                                              #
################################################################################


REPO_URI=https://github.com/titepweb/nmdf
REPO_BRANCH=master
APP_URL=https://raw.githubusercontent.com/titepweb/nmdf/master/bin/nmdf


# @todo: use variable instead of value
app_name=nmdf
_symlink=.symlink
_copy=.copy


#### A MANUAL AS WELL AS A HELP FUNCTION ######################################


function help {
cat << EOF
--------------------------------------------------------------------------------
                        THE WORKFLOW OF USING nmdf


  (1) Update/Clone dotfiles : $(basename $0) -u [-r repository] [-d path to dir]


    If run from remote location, nmdf automatically clone the REPO_URI
    [$REPO_URI]
    to the working directory or the specified directory that user give through
    the option -d {path to dir}. -d would be useful to manage multiple dotfiles.


  (2) Install dotfiles      : $(basename $0) command [options]
      * command :
        -l  : Symlink both *.copy and *.symlink  files
        -c  : Copy both *.copy and *.symlink files
        -i  : Install : Copy *.copy files and symlink *.symlink files accordinly.
              Skip copying if destination file exists.
        -if : Reinstall forcefully : Install dotfiles, do not skip copying.


      * options :
        -o  : Create symlinks without overwrite confirmation.
        -b  : Create symlinks without backup confirmation.
        -a  : Ask user for copying each file or creating each symlink.


  (3) Show statistic data about dotfiles : $(basename $0) -s


  (4) Push dotfiles to VSC  : $(basename $0) [-r repo] -p ["commitMessage"]


      If the commitMessage is not specified or too short, an auto-message will
      be generated.


  (*) Tips: $(basename $0) -uiba
--------------------------------------------------------------------------------
                      The CORE CHARACTERISTICS of nmdf


  1. flexibility : cross-platform (not tested yet)


  2. modularity : topic-central organization


  3. centralization


  4. support copy and symlink option. Why? First and foremost, certain dotfiles
     could contains sensitive information, as in muttrc (email password), ssh
     keys, ... , so it need to copied and manually configured; a prompt is nice.
     Secondly, some softwares, like newsbeuter, does not accept configuration
     symlink files. Thirdly, we need one-way changes from our dotDir to local
     configuration files for some programs; therefore, we will not be out of
     controlled. Futhermore, we could nmdf -l to use linking instead of copying
     and nmdf -c à la the well-known Stow tool.


  5. smart storing : Unlike the alternative scripts which store .dotfiles and
     their ~/.data in dorDir with .symlink, nmdf only stores what really matter.
     When installing, nmdf deeply scan for *.symlink and *.copy files and
     directories in every topic. Therefore, nmdf keep dotDir lightweight due to
     linking and copying only essential files instead of the whole directories
     which may contain overweight temp files, for instance, ~/.config/chromium.
```

```
     For example, do not use ~/.config/sublime-text-3.symlink. We just need to
     symlink 2 files, then install PackageControl, and let it take over the rest.
     + ~/.config/sublime-text-3/Packages/User/Package Control.sublime-settings
     + ~/.config/sublime-text-3/Packages/User/Package Control.sublime-settings


     Another example: For vim, we just need to symlink ~/.vimrc, install vundle,
     run :PlugInstall in vim or 'vim +PluginInstall +qall' from command line to
     install/update vim plugins.


     NOTE: 'git submodule' is briliant to manage pluggins of other programs!


  6. including pull for updates from and push changes to the REPO_URI.


  7. automatically decide git clone/pull based on whether nmdf is run remotely.


--------------------------------------------------------------------------------
        The ORAGNIZATION MODEL of DOTFILES UNDER the MANAGEMENT of nmdf


 dotfiles                 # topic-centric organization dotfiles
 ├─ bin  (required)       # added to your \$PATH and be made available everywhere
 │   ├─ nmdf              # nmdf should be here for easily often running.
 │   ├─ xxx               # xxx is used to test new learned bash codes.
 │   └─ ...               #
 ├─ temp (optional)       # --> add this folder into .gitignore
 ├─ nmdf.log              # --> add this folder into .gitignore
 ├─ example               #
 │   ├─ nmExplample.md    # --> a decent place to type your discovering process...
 │   ├─ related files ...  #
 │   ├─ ...
 │   ├─ .exclude-platforms # nmdf #1: specifies excluded platforms for this topic
 │   ├─ example.symlink    # nmdf #2a: symlinked to ~/.example if ...
 │   ├─ example.copy       # nmdf #2b: copied to ~/.example if ...
 │   ├─ .install.sh        # nmdf #3: performs additional installation after symlinking
 │   ├─ zprofile.zsh       # zsh #1: run for login shells
 │   ├─ path.zsh           # zsh #2: modifies \$PATH
 │   ├─ something.zsh      # zsh #3: additional setup
 │   ├─ postinit.zsh       # zsh #4: run after additional setup
 │   └─ completion.zsh     # zsh #5: zsh completion setup
 └─ git
     ├─ bin                # contains git scripts
     │   └─ git-specific-script
     ├─ gitconfig.symlink  # symlinked to ~/.gitconfig
     ├─ aliases.zsh        # sets up git aliases
     ├─ path.zsh           # adds dotfiles/git/bin to \$PATH
     └─ ...
--------------------------------------------------------------------------------
        ************************************************************
        *                       WARNING:                           *
        *    Make sure that your sensitive data (SSH keys, ...)    *
        *              are not included in dotfiles.               *
        ************************************************************
EOF
exit 0
}


#### LIBRARIES ALLOWING nmdf TO RUN FROM A REMOTE LOCATION ###################


function updateRepo {
  # Check if git is installed before doing git operations
  type git >/dev/null 2>&1 || {
    echo "************************************"
    echo >&2 "Git is required to clone/pull/push dot files from/to a remote repository. "\
    "Symlinks can still be auto created to files in $dotDir but a remote repository will not be used.";
    echo "************************************"
    return;
  }


  # if @1="pull" but there is no repo here create one, otherwise update it afterwards.
  if [ "$1" == "pull" -a ! -d "$dotDir/.git" ]; then
    displayTitle "DOWNLOADING MASTER REPOSITORY"
    [ -e "$dotDir"] && fail "${red}${bold}$dotDir is a regular file or a not empty directory. 'git clone' only work on empty
directory.${reset}"
    git clone -b "$REPO_BRANCH" "$REPO_URI" "$dotDir"
    success "${green}${bold}Cloning $app_name completed!${reset}"
    return
  fi


  # If dotDir does not exist, return.
  [ ! -d "$dotDir" ] && fail "${red}${bold}The dotDir $dotDir does not exist. \nUpdating Repository fails!${reset}"


  # Git Pull or Push
  if [ "$1" == "pull" ]; then
```

```
    displayTitle "Pulling changes from the $app_name's repository, branch $REPO_BRANCH"
    (cd $dotDir && git pull origin "$REPO_BRANCH")
    success "${green}${bold}\nPulling changes from the $app_name completed!${reset}"
  elif [ "$1" == "push" ]; then
    displayTitle "Pushing changes to the $app_name's repository, branch $REPO_BRANCH"
    cd $dotDir && git add . && git commit -m "$commitMessage" && git push origin "$REPO_BRANCH"
    echo -e "${green}${bold}\nPushing changes from the $app_name repository completed!${reset}"
  fi
}


#### LIBRARIES OFFERING INFORMATIVENESS #####################################


### COLOR DEFENITION ### reference: https://en.wikipedia.org/wiki/ANSI_escape_code
# Note: \033 = \e   30+i to specify foreground color, 40+i to specify background color,
reset='\e[0m'     ;


bold='\e[1m'      ;   faint='\e[2m'       ;   italic='\e[3m'       ;   underline='\e[4m'    ;
blink='\e[5m'     ;   blinkfast='\e[6m'   ;   negative='\e[7m'     ;   conceal='\e[8m'      ;
strike='\e[9m'    ;


normal='\e[22m'   ;   roman='\e[23m'      ;   nounderline='\e[24m' ;   noblink='\e[25m'     ;
positive='\e[27m' ;   reveal='\e[28m'     ;   nostrike='\e[29m'    ;


black='\e[30m'    ;   red='\e[31m'        ;   green='\e[32m'       ;   yellow='\e[33m'      ;
blue='\e[34m'     ;   magenta='\e[35m'    ;   cyan='\e[36m'        ;   white='\e[37m'       ;
xterm='\e[38'     ;   default='\e[39m'    ;


bgblack='\e[40m'  ;   bgred='\e[41m'      ;   bggreen='\e[42m'     ;   bgyellow='\e[43m'    ;
bgblue='\e[44m'   ;   bgmagenta='\e[45m'  ;   bgcyan='\e[46m'      ;   bgwhite='\e[47m'     ;
bgxterm='\e[48'   ;   bgdefault='\e[49m'  ;
# USAGE : printf , echo -e "just a ${underline}${red}${bold}underline red${reset}, ${negative}NEGATIVE${reset}, and
${strike}bold${reset} text."


# Fail on errors.
set -e
#set -x


# COLORFULLY INFORM ✱╳—•…›✓
# NOTE: echo -e is preffered because of avoiding using \n at the end of the command
scan () {
  echo -e "\r  [${bold}${magenta} SCAN ${reset}] ${magenta}$1${reset}"
}
info () {
  echo -e "\r  [${bold}${cyan} INFO ${reset}] $1"
}
user () {
  echo -e "\r  [ ${bold}${yellow} ?? ${reset} ] $1"
}
success () {
  printf "\r  [ ${bold}${green} OK ${reset} ] $1\n"
}
fail () {
  printf "\r  [${bold}${red} FAIL ${reset}] $1\n"
  exit 1
}


function displayTitle {
  #NOTE: [01;04;33  = bold + underline + xterm
  BoldUnderlinedRed='\n\e[01;04;38m%s\e[0m\n'
  printf $BoldUnderlinedRed "$1"
}


function stats {
  cd $dotDir  # cd to dotdir before du -sh and git add .
  displayTitle "STATISTIC DATA"


  echo -e "-dotfiles in     :${blue}${bold} $dotDir${reset}"
  if [ -e "$dotDirIns" ]; then
    local insState="$(readsymlink "$dotDirIns")"
    if [ "$insState" == "$dotDir" ]; then
      echo -e "-installed in    :${blue}${bold} $dotDirIns${reset}"
    fi
  fi
  if [ ! -e "$dotDirIns" ] || [ -z "$insState" ]; then
    echo -e "-dotDir symlink  :${red}${bold} not installed${reset}"
  fi
  echo -e "-total files     :${yellow}${bold} $(find $dotDir -type f | wc -l)${reset} \t files"
  echo -e "-total directory :${yellow}${bold} $(find $dotDir -type d | wc -l)${reset} \t directories"
  echo -e "-usage           :${red}${bold} $(du -sh)${reset}"
  echo -e "-dotfiles        :${cyan}${bold} $(find $dotDir -type f \
    -not -path "*/.git/*" \
    -not -path "*/.hg/*" \
```

```
    -not -path "$tempDir/*" \
    -not -path "$backupDir/*" \
    | wc -l)${reset} \t files"
  echo -e "-dotdirectories  :${cyan}${bold} $(find $dotDir -type d \
    -not -path "*/.git/*" \
    -not -path "*/.hg/*" \
    -not -path "$tempDir/*" \
    -not -path "$backupDir/*" \
    | wc -l)${reset} \t directories"
  echo "----------------------------------------"
  git add . && git status
  exit 0
}


#### LIBRARIES OFFERING PORTABILITY ##########################################


detectplatform () {
  if [ -n "$platform_cache" ]; then
    printf "$platform_cache"
    return
  fi


  if [ "$(uname)" == "Darwin" ]; then
    printf 'mac'
  elif [ "$(uname -s)" == "Linux" ]; then
    printf 'linux'
  elif [ "$(uname -s)" == "FreeBSD" ]; then
    printf 'freebsd'
  elif [ "$(uname -o)" == "Cygwin" ]; then
    printf "cygwin\nwindows"
  elif [ "$(uname -o)" == "Msys" ]; then
    printf 'msys'
  else
    printf 'unknown'
  fi
}
# Cache value to avoid repeating command "uname"
export platform_cache="$(detectplatform)"


readsymlink () {
  local file=$1


  if [ "$(detectplatform)" == 'freebsd' ]; then
    printf "$(realpath "$file")"
    return $?
  fi


  printf "$(readlink --canonicalize-existing "$file")"
}


detecthome () {
  local platform=$1


  case "$platform" in
    mac|linux|msys|cygwin|freebsd )
      home=$HOME
      ;;
    windows )
      home="$(printf "%s" "$(cygpath --unix $USERPROFILE)")"
      ;;
    * )
      fail "Cannot determine home directories for platform ${red}$platform${reset}"
      ;;
  esac
}


platform_excluded () {
  local topic=$1
  local platform=$2
  local exclude_file="$topic/.exclude-platforms"


  if [ ! -f "$exclude_file" ]; then
    return 1
  fi


  if [ -n "$(/usr/bin/grep "$platform" "$exclude_file")" ]; then
    return 0
  fi


  return 1
}
```

```
#### THE CORE LIBRARIES FOR nmdf ##############################################


MkParentFolder() {
  local inDir="$(dirname "$1")"

  # Remove $inDir if it is a broken symbolic file
  if [[ -h "$inDir" ]]; then
    [[ -z "$(readsymlink "$inDir" )" ]] && rm "$inDir"
  # ask for remove $inDir if it is a regular file
  elif [[ -f "$inDir" ]]; then
    info "File already exists: ${cyan}$inDir${reset}"
    user "Delete it? [${green}${bold}y${reset}/${red}${bold}n${reset}"]
    read -n 1 action < /dev/tty
    # if not yes, then skip it
    if [[ $action =~ ^[Yy]$ ]]; then
      rm "$inDir"
      success "Removed${red} ${strike}$inDir${reset}"
    else
      fail "Take a look at ${red}$inDir${reset}"
    fi
  fi


  # Create $inDir if it does not exist
  if [ ! -d "$inDir" ]; then
    mkdir -p "$inDir"       # mkdir can make two parent & son directories just with mkdir /parent_folder/son_folder
    success "Created $inDir"
  fi
}


SymlinkOrCopy () {
  local src=$1
  local dst=$2
  local ext="${src##*.}"


  local overwrite= ; local backup= ; local skip=
  local action=


  # Choose the right word to ask ---------------------------------------------
  if ( [ "$src" == "$dotDir" ] ) || ( [ "$installOption" == "install" ] && [ "$ext" = "symlink" ] ) || ( [ "$installOption" ==
"symlink" ] ); then
      local askword1="linked to"
      local askword2="Create symlink"
  elif ( [ "$installOption" == "install" ] && [ "$ext" == "copy" ] ) || ( [ "$installOption" == "copy" ] ); then
      local askword1="overwrited by"
      local askword2="Copy"
  fi


  # Ask user if destination file exists --------------------------------------
  if [ -f "$dst" -o -d "$dst" -o -L "$dst" ]; then
    if [ "$overwrite_all" == "false" ] && [ "$backup_all" == "false" ] && [ "$skip_all" == "false" ]; then
      local current_src="$(readsymlink "$dst")"


      if [ "$installOption" == "install" ] && [ "$ext" == "copy" ] && [ "$forcefullyOption" != "true" ]; then
        skip=true;
      elif [ "$current_src" == "$src" ]; then
        skip=true;
      else
        user "File already exists: ${green}$dst${reset} \n\t   (${cyan}$(file --brief "$dst")${reset})
           Will be $askword1 ${green}$src${reset}
           ${bold}${yellow}s${reset}kip, ${bold}${yellow}S${reset}kip all, ${bold}${yellow}o${reset}verwrite,
${bold}${yellow}O${reset}verwrite all, ${bold}${yellow}b${reset}ackup, ${bold}${yellow}B${reset}ackup all?"


        while true; do
          # Read from tty, needed because we read in outer loop.
          read -n 1 action < /dev/tty


          case "$action" in
            o ) overwrite=true     ; break ;;
            O ) overwrite_all=true ; break ;;
            b ) backup=true        ; break ;;
            B ) backup_all=true    ; break ;;
            s ) skip=true          ; break ;;
            S ) skip_all=true      ; break ;;
            * ) ;;
          esac
        done
      fi
    fi


    overwrite=${overwrite:-$overwrite_all}
    backup=${backup:-$backup_all}
    skip=${skip:-$skip_all}
```

```
    if [ ! "$skip" == true ] && [ ! "$dst" == "$dotDir" ]; then
      if [ "$overwrite" == "true" ]; then
        rm -rf "$dst"
        success "Removed ${green}$dst${reset}"
      fi


      if [ "$backup" == "true" ]; then
        # Back $dst up without loosing directory structure
        local relativeFile=${dst#$home} #remove $home from $file
        # Remove the leading slash if there is one
        if [[ $relativeFile == /* ]]; then
          relativeFile=${relativeFile:1}
        fi
        local bakfile="$backupDir/$relativeFile.BAK.$(date +"%Y-%m-%d_%H-%M")"
        # Make sure that the destination directory exists before backing up.
        MkParentFolder "$bakfile"
        # Backup
        mv "$dst" "$bakfile"
        success "${green}┌─${reset}Moved ${green}$dst${reset} \n \t   ${green}└─ ✓ ─»${reset} ${green}$bakfile${reset}"
      fi
    fi
  fi


  # Skip , Symlink , or Copy ------------------------------------------------
  if [ "$skip" == "true" ]; then
    success "Skipped┌${green}$src${reset} \n \t   ${red}   X   ${reset}└${green}$dst${reset}"
  fi

  if [ "$skip" != "true" ]; then  # "false" or empty
    # Make sure that the destination directory exists before creating symlinks or copying
    MkParentFolder "$dst"
    # Ask users before creating symlinks or copying
    if [ "$askOption" == "true" -a "$overwrite" != "true" -a "$backup" != "true" ]; then
      user "$askword2 $(dirname "$dst")/${yellow}$(basename "$dst")${reset}? [${green}${bold}y${reset}/${red}${bold}n${reset}/install
${green}${bold}a${reset}ll] "
      read -n 1 action < /dev/tty
      # if not yes, then skip it
      if [[ $action =~ [Aa] ]]; then
        askOption=false
      elif [[ ! $action =~ [Yy] ]]; then
        success "Skipped┌${green}$src${reset} \n \t   ${red}   X   ${reset}└${green}$dst${reset}"
        return
      fi
    fi


    # Create symlink or copy
    if ( [ "$src" == "$dotDir" ] ) || ( [ "$installOption" == "install" ] && [ "$ext" = "symlink" ] ) || ( [ "$installOption" ==
"symlink" ] ); then


      # Create native symlinks on Windows.
      export CYGWIN=winsymlinks:nativestrict


      # Symlink command
      ln -s "$src" "$dst"
      symlinkindex=$(($symlinkindex+1))
      success "${green}╔Linked=${reset}${green}$src${reset} \n \t   ${green}╚═ ✓ ══${reset}${green}$dst${reset}"


    elif ( [ "$installOption" == "install" ] && [ "$ext" == "copy" ] ) || ( [ "$installOption" == "copy" ] ); then


      # Copy command (-r stands for recursively copy )
      cp -r "$src" "$dst"
      copyindex=$(($copyindex+1))
      success "${green}╔Copied=${reset}${green}$src${reset} \n \t   ${green}╚═ ✓ ==>${reset}${green}$dst${reset}"


    fi
  fi
}


function RemoveTailing () {
  for i in `seq 1 10`; #10 here is customizable
    do
      for file in $( \
        find "$1" \
          -mindepth $i -maxdepth $i \
          -not -path "*/*.ignore/*" \
          -not -path "*/80%/*" \
          -not -path "$tempDir/*" \
          -name '*.symlink' -or -name '*.copy' \
        \
         ); do
        if [[ $file =~ .*\.(symlink|copy)/.*\.(symlink|copy)$ ]]; then
          # remove the tailing .symlink or .copy
          local renameto="${file%.*}"
```

```
          local ext="${file##*.}"
          mv "$file" "$renameto"
          fixnameindex=$(($fixnameindex+1))
          # highlight .symlink in the string $file by replacing matched '.symlink'
          renameto=${renameto/.symlink/${red}.symlink${reset}}
          renameto=${renameto/.copy/${red}.copy${reset}}
          echo -e "  $renameto${strike}${red}.$ext${reset}"
        fi
      done
    done
}


#### LOOP THROUGH FILES #########################################################


install_dotfiles () {
  displayTitle "INSTALLING DOTFILES"
  info "Installing dotfiles from ${cyan}$dotDir${reset}\n"


  local scanindex=0;
  local fixnameindex=0;
  local symlinkindex=0;
  local copyindex=0;


  # local platforms="$(detectplatform)"
  # while IFS=$'\n' read -r platform; do
  #   local home
  #   detecthome "$platform"

    info "Installing dotfiles in ${cyan}$home${reset} for platform ${cyan}$platform${reset}"
    # First, add a symlink for this dotfiles directory.
    # if [[ ! "$dotDir" == "$dotDirIns" ]]; then
      SymlinkOrCopy "$dotDir" "$dotDirIns"
    # fi


    # Find direct child directories (topics), exclude those starting with dots.
    local topics="$(/usr/bin/find "$dotDir" -mindepth 1 -maxdepth 1 -type d -not -name '\.*' -not -path '*/*.ignore/*' )"
    while IFS=$'\n' read -r topic; do
      [[ -z "$topic" ]] && continue


      scan "-Topic- $topic"


      if platform_excluded "$topic" "$platform"; then
        info "Skipped ${green}$topic${reset} as it is excluded for platform ${cyan}$platform${reset}"
        continue
      fi


      # Remove the taling .symlink in the names of all the sub-directories of another *.symlink directory
      RemoveTailing "$topic"


      # Find files and directories named *.symlink below each topic directory, exclude files in 80% directories.
      # local symlinks="$(/usr/bin/find "$topic" -mindepth 1 -maxdepth 1 \( -type f -or -type d \) -name '*.symlink')"
      local symlinks="$(/usr/bin/find "$topic" -mindepth 1 \
        -not -path "*/*.ignore/*" \
        -not -path "*/80%/*" \
        -not -path "$tempDir/*" \
        -and \
        \( \
          \(   -type f \
               -name '*.symlink' -or -name '*.copy' \
          \) -or \( \
               -type d \
               -name '*.symlink' -or -name '*.copy' \
          \) \
        \) \
        )"


      while IFS=$'\n' read -r src; do
        [[ -z "$src" ]] && continue


        # remove the leading $topic from the destination directory name
        local dst="$home/${src#$topic/}"
        # remove the leading .symlink or .copy from the destination directory name
        dst=${dst%.symlink}
        dst=${dst%.copy}


        SymlinkOrCopy "$src" "$dst"
        scanindex=$(($scanindex+1))


      done <<< "$symlinks"
```

```
      # Run optional install script.
      # 1. remember to set execute permissions chmod +x <script> for the .install.sh scripts. Otherwise, nmdf skip running it.
      # 2. make sure that the script doesn't stop nmdf to continue scanning topic and installing dotfiles
      local install="$topic/.install.sh"
      if [ -x "$install" ]; then
        info "Running ${green}$install${reset}"
        sh -c "$install"
      fi
    done <<< "$topics"


  # done <<< "$platforms"


  echo
  info  "${cyan}${bold}$scanindex${reset} dotfile(s) are scanned from ${cyan}$dotDir${reset}"
  info  "${red}${bold}$fixnameindex${reset} dotfile(s)'s name(s) are fixed '${strike}${red}.symlink${reset} &
${strike}${red}.copy${reset}'!"
  info  "${cyan}${bold}$symlinkindex${reset} dotfile(s) are symlinked!"
  info  "${cyan}${bold}$copyindex${reset} dotfile(s) are copied!"
  info  "${cyan}${bold}$(($symlinkindex + $copyindex))${reset} dotfile(s) are installed, totally!"
}


#### MAIN #####################################################################


# STEP 1 # Parse option arguments or ask for them if not been specified
# ---------------------------------------------------------------------
  function askParameters {
    displayTitle "WHAT WOULD YOU LIKE TO LAUNCH ?"
    PS3="Select above option regarding how to create the symlinks:"
    options=("Update/Clone from $REPO_URI"
    "Install dotfiles"
    "Rinnstall dotfiles"
    "Install dotfiles. Ask before creating each symlink"
    "Reinstall dotfiles. Ask before creating each symlink"
    "Show statistic data"
    "Push changes to $REPO_URI"
     )
    select opt in "${options[@]}"
    do
      case $opt in
        "Update/Clone from $REPO_URI")                        pullOption=true ; break ;;
        "Install dotfiles")                                   installOption="install" ; break ;;
        "Reinstall dotfiles")                                 installOption="install" ; forcefullyOption=true ; break ;;
        "Install dotfiles. Ask before creating each symlink")   askOption=true ; installOption="install" ; break ;;
        "Reinstall dotfiles. Ask before creating each symlink") askOption=true ; installOption="install" ; forcefullyOption=true ;
break ;;
        "Show statistic data")                                showStats=true ; break ;;
        "Push changes to $REPO_URI")                          pushOption=true ; break ;;
      *) fail "invalid option" ;;
      esac
    done
  }


  # Default Options
  overwrite_all=false backup_all=false skip_all=false


  if [ $# == 0 ]; then
    askParameters
  else
    [ "$1" ==  "--help" ] && help
    OPTIND=1 # Reset in case getopts has been used previously in the shell.
    #NOTE: Never use getopt (Traditional versions of getopt cannot handle empty argument strings, or arguments with embedded
whitespace.)
    while getopts :uclifaobsd:r:p: OPTION; do
      case $OPTION in
        u)  pullOption=true ;;
        c)  installOption="copy" ;;
        l)  installOption="symlink" ;;
        i)  installOption="install" ;;
        f)  forcefullyOption=true ;;
        a)  askOption=true ;;
        o)  overwrite_all=true ;;
        b)  backup_all=true ;;
        s)  showStats=true ;;
        d)  dotDir=$OPTARG ;;
        r)  REPO_URI=$OPTARG ;;
        p)  pushOption=true ;
            commitMessage=$OPTARG
            [[ -z "$commitMessage" ]] || [[ ${#commitMessage} -lt 4 ]] && commitMessage="Auto commit dot files at $(date)"
        ;;
        :)  #@FIXME: This option is not considered if p,d,r option is follow by another argument; for example, nmdf -dr , nmdf -pa. So,
nmdf -ap is desired instead of nmdf -pa
            if [ "$OPTARG" == "p" -a -z "${commitMessage// }" ]; then
              commitMessage="Auto commit dot files at $(date)"
              echo "Automatic commit message :"$commitMessage
            else
              echo "Option -$OPTARG requires an argument. Type nmdf -h for usage."
              exit 1
            fi
```

```
        ;;
        h|\?) # if unrecognized option - show help
            help
            exit 1
        ;;
      esac
    done
    shift $((OPTIND-1))  #This tells getopts to move on to the next argument.
  fi


# STEP 2 # Check Platform and Create global variables
# -------------------------------------------------
  platform="$(detectplatform)"
  detecthome "$platform"    # create the global variable home


  backupDir=$home/dotfiles-backup     #--> This folder should not be in $dotDir because if we delete $dotDir, we also delete $backupDir
  # dotDir will be used to create symlink from $dotDir to $dotDir
  dotDirIns=$home/.dotfiles           #--> Don't change this because bashrc, ... use the path ~/.dotfiles


  # If user didn't provide -d path/to/directory, ...
  if [ -z "$dotDir" ]; then
    # a modified version from : http://stackoverflow.com/questions/59895/can-a-bash-script-tell-what-directory-its-stored-
in/246128#246128
    SOURCE="${BASH_SOURCE[0]}"
    while [ -h "$SOURCE" ]; do # resolve $SOURCE until the file is no longer a symlink
      DIR="$( cd -P "$( dirname "$SOURCE" )" && pwd )"
      SOURCE="$(readsymlink "$SOURCE")"
      [[ $SOURCE != /* ]] && SOURCE="$DIR/$SOURCE" # if $SOURCE was a relative symlink, we need to resolve it relative to the path
where the symlink file was located
    done
    dotDir="$( cd -P "$( dirname "$SOURCE" )/.." && pwd )"  #--> nmdf must be in $binDir, not in $dotDir
  fi


  # Trim the trailing slash from the dot directory if user mistyped.
  [[ $dotDir == */ ]] && dotDir="${dotDir%?}"  # OR dotDir=${dotDir:1}


  # Set special directories and files based on defined dotDir.
  binDir=$dotDir/bin                  #--> include in .bashrc or .bash_profile: export PATH=$binDir:$PATH
  tempDir=$dotDir/temp                #--> include in .gitignore


  # currently, logfile is not used. Save these for the later version of nmdf
  logfile=$dotDir/nmdf.log            #--> include in .gitignore
    # Delete log file if its size is bigger than 50 KiB
    if [ -f $logfile ]; then
      size=$(stat -c%s $logfile)
      [[ $size > 50000 ]] && rm $logfile
    fi


  # Check the global variagles before actions
  function Debug () {
    displayTitle "THE CORE PARAMETERS"
    echo "-pullOption    =" $pullOption
    echo "-installOption =" $installOption
    echo "-overwrite_all =" $overwrite_all
    echo "-backup_all    =" $backup_all
    echo "-skip_all      =" $skip_all
    echo "-askOption     =" $askOption
    echo "-showStats     =" $showStats
    echo "-dotDir        =" $dotDir
    echo "-REPO_URI      =" $REPO_URI
    echo "-REPO_BRANCH   =" $REPO_BRANCH
    echo "-pushOption    =" $pushOption
    echo "-commitMessage =" $commitMessage
    echo "-----------------------------------------"
  }


# STEP 3 # - Create a working copy of the dofiles repository.
#          - Pull changes if dotfiles repository exists.
#          - Evaluate the default parameters or what user typed (-d directory).
#--------------------------------------------------------------------------------
  if [ "$pullOption" == "true" ]; then
    updateRepo "pull"
  fi


  # @FIXME:
  # sh -c "`curl -fsSL https://raw.githubusercontent.com/titepweb/nmdf/master/bin/nmdf`"
  # sh -c "$(wget https://raw.githubusercontent.com/titepweb/nmdf/master/bin/nmdf -O -)"
  # bash <(wget -nv -O -https://raw.githubusercontent.com/titepweb/nmdf/master/bin/nmdf)
  # if this script is running from a remote location and the dotfiles directory was empty, download and save nmdf to the bin directory,
however,
  if [ ! -f "$binDir/nmdf" ] && [ ! hash nmdf >/dev/null 2>&1 ]; then
    curl -fsSL -o "$binDir/nmdf" $APP_URL
    chmod +x "$binDir/nmdf"
  fi
```

```
  # After pulling, dotDir must be created or filled with content. So, Quit if dotDir does not exist.
  [ ! -d "$dotDir" ] && fail "Dotfile directory ${red}'$dotDir'${reset} does not exist. nmdf is terminated."


  # Verify that the backup directory exists
  [ ! -d "$backupDir" ] && mkdir -v -p "$backupDir"


  # Create special directory parameters:
  [ ! -d "$binDir" ] && mkdir -v -p "$binDir"
  [ ! -d "$tempDir" ] && mkdir -v -p "$tempDir"


# STEP 4 # - Create ContextualSymlinkList which is used in SymlinkOrCopy functions.
#          - Rename symlink sub-directories of another symlink directories.
#          - Install dotfiles by creating symlinks. Backup existed dotfiles if necessary.
#-------------------------------------------------------------------------------------------
  if [ -n "$installOption" ]; then
    # for the first time, submodule need to be
    displayTitle "INITIALIZING SUBMODULES"
    cd "$dotDir" && git submodule update --init --recursive #--recrusive means update all submodules within
    info "Completed!"
    # ContextualSymlinkList
    install_dotfiles
  fi


  if [ "$showStats" == "true" ]; then
    stats
  fi
# STEP 5 # Push changes to the nmdf repository, which is useful if nmdf renamed/created directories
#-----------------------------------------------------------------------------------------------------
  if [ "$pushOption" == "true" ]; then
    updateRepo "push"
  Fi

exit 0
#### FUNCTION BANK ############################################################


  # RemoveTailing function can be rewritten :
  for i in `seq 1 10`; #10 here is customizable
    do
      find "$1" -mindepth $i -maxdepth $i -name '*.symlink' -exec sh -c ' {
        if [[ $0 =~ .*\.symlink/.*\.symlink$ ]]; then
          mv "$0" "${0%.symlink}"
          count_rename=$(($count_rename+1))
          echo -e "  [\e[31mrename\e[0m] ${0%.symlink}\e[9;31m.symlink\e[0m"
        fi
      }' {} \;
    done
          -not -path "$tempDir/*" \


  # Before I find out mkdir can make multi-level directories, I use this to test if a directory contains .symlink file(s) or not.
  # If a directory has a *.symlink file inside, I use the following codes to make sure that the destination directory exists before
creating the corresponding symlinks.
  if [ -d "$src" ]; then
    local results=( $(find "$src" -name "*.symlink") )
    if (( ${#results[@]} )) ; then
      if [[ -d "$dst" ]]; then
        scan "Existed $dst"
      else
        mkdir -p "$dst"
        success "Created $dst"
      fi
    else
      scan "Not found any *.symlink files in $src"
    fi
  fi
```