

zsh configuration

`zsh` configuration as provided on a standard Manjaro install with some additions

General settings

```
## Options section
setopt correct # Auto correct mistakes
setopt extendedglob # Extended globbing. Allows
using regular expressions with *
setopt nocaseglob # Case insensitive globbing
setopt rcexpandparam # Array expansion with
parameters
setopt nocheckjobs # Don't warn about running
processes when exiting
setopt numericglobsort # Sort filenames numerically
when it makes sense
setopt nobeep # No beep
setopt appendhistory # Immediately append history
instead of overwriting
setopt histignorealldups # If a new command is a
duplicate, remove the older one
setopt autocd # if only directory path is
entered, cd there.
setopt inc_append_history # save commands are added to
the history immediately, otherwise only when shell exits.

zstyle ':completion:*' matcher-list 'm:{a-zA-Z}={A-Za-z}' # Case insensitive tab
completion
zstyle ':completion:*' list-colors "${(s..)LS_COLORS}" # Colored completion
(different colors for dirs/files/etc)
zstyle ':completion:*' rehash true # automatically find new
executables in path
# Speed up completions
```

```

zstyle ':completion:*' accept-exact '*(N)'
zstyle ':completion:*' use-cache on
zstyle ':completion:*' cache-path ~/.zsh/cache
HISTFILE=~/.zhistory
HISTSIZ=10000
SAVEHIST=10000
#export EDITOR=/usr/bin/nano
#export VISUAL=/usr/bin/nano
WORDCHARS=${WORDCHARS//\/[&.]} # Don't consider certain
characters part of the word

```

Key Bindings

```

## Keybindings section
bindkey -e
bindkey '^[[7~' beginning-of-line # Home key
bindkey '^[[H' beginning-of-line # Home key
if [[ "${terminfo[khome]}" != "" ]]; then
    bindkey "${terminfo[khome]}" beginning-of-line # [Home] - Go to beginning of
line
fi
bindkey '^[[8~' end-of-line # End key
bindkey '^[[F' end-of-line # End key
if [[ "${terminfo[kend]}" != "" ]]; then
    bindkey "${terminfo[kend]}" end-of-line # [End] - Go to end of line
fi
bindkey '^[[2~' overwrite-mode # Insert key
bindkey '^[[3~' delete-char # Delete key
bindkey '^[[C' forward-char # Right key
bindkey '^[[D' backward-char # Left key
bindkey '^[[5~' history-beginning-search-backward # Page up key
bindkey '^[[6~' history-beginning-search-forward # Page down key

# Navigate words with ctrl+arrow keys
bindkey '^[[Oc' forward-word #
bindkey '^[[Od' backward-word #
bindkey '^[[1;5D' backward-word #

```

```
bindkey '^[[1;5C' forward-word #
bindkey '^H' backward-kill-word # delete previous word with
ctrl+backspace
bindkey '^[[Z' undo # Shift+tab undo last action
```

Aliases

```
## Alias section
alias ls='ls --color=auto'
alias ll='ls -lahF'
alias cp='cp -i' # Confirm before overwriting
something
alias df='df -h' # Human-readable sizes
alias free='free -m' # Show sizes in MB
alias gitu='git add . && git commit && git push'
```

Theming

```
# Theming section
autoload -U compinit colors zcalc
compinit -d
colors

# Color man pages
export LESS_TERMCAP_mb=$'\E[01;32m'
export LESS_TERMCAP_md=$'\E[01;32m'
export LESS_TERMCAP_me=$'\E[0m'
export LESS_TERMCAP_se=$'\E[0m'
export LESS_TERMCAP_so=$'\E[01;47;34m'
export LESS_TERMCAP_ue=$'\E[0m'
export LESS_TERMCAP_us=$'\E[01;36m'
export LESS=-R
```

Plugins

```
## Plugins section: Enable fish style features
# Use syntax highlighting
source /usr/share/zsh/plugins/zsh-syntax-highlighting/zsh-syntax-highlighting.zsh

# Use history substring search
source /usr/share/zsh/plugins/zsh-history-substring-search/zsh-history-substring-search.zsh

# bind UP and DOWN arrow keys to history substring search
zmodload zsh/terminfo
bindkey "$terminfo[kcuu1]" history-substring-search-up
bindkey "$terminfo[kcud1]" history-substring-search-down
bindkey '^[[A' history-substring-search-up
bindkey '^[[B' history-substring-search-down

source /usr/share/zsh/plugins/zsh-autosuggestions/zsh-autosuggestions.zsh
ZSH_AUTOSUGGEST_BUFFER_MAX_SIZE=20
ZSH_AUTOSUGGEST_HIGHLIGHT_STYLE='fg=8'
```

Terminal Window Title

```
# Set terminal window and tab/icon title
#
# usage: title short_tab_title [long_window_title]
#
# See: http://www.faqs.org/docs/Linux-mini/Xterm-Title.html#ss3.1
# Fully supports screen and probably most modern xterm and rxvt
# (In screen, only short_tab_title is used)
function title {
    emulate -L zsh
    setopt prompt_subst

    [[ "$EMACS" == *term* ]] && return

    # if $2 is unset use $1 as default
    # if it is set and empty, leave it as is
    : ${2=$1}
```

```

case "$TERM" in
  xterm*|putty*|rxvt*|konsole*|ansi|mlterm*|alacritty|st*)
    print -Pn "\e]2;${2:q}\a" # set window name
    print -Pn "\e]1;${1:q}\a" # set tab name
    ;;
  screen*|tmux*)
    print -Pn "\ek${1:q}\e\\" # set screen hardstatus
    ;;
  *)
    # Try to use terminfo to set the title
    # If the feature is available set title
    if [[ -n "$terminfo[fsl]" ]] && [[ -n "$terminfo[tsl]" ]]; then
      echoti tsl
      print -Pn "$1"
      echoti fsl
    fi
    ;;
esac
}

ZSH_THEME_TERM_TAB_TITLE_IDLE="%15<..<%~%<<" #15 char left truncated PWD
ZSH_THEME_TERM_TITLE_IDLE="%n@m:%~"

# Runs before showing the prompt
function mzc_termsupport_precmd {
  [[ "${DISABLE_AUTO_TITLE:-}" == true ]] && return
  title $ZSH_THEME_TERM_TAB_TITLE_IDLE $ZSH_THEME_TERM_TITLE_IDLE
}

# Runs before executing the command
function mzc_termsupport_preexec {
  [[ "${DISABLE_AUTO_TITLE:-}" == true ]] && return

  emulate -L zsh

  # split command into array of arguments
  local -a cmdargs
  cmdargs=("${(z)2}")
  # if running fg, extract the command from the job description

```

```

if [[ "${cmdargs[1]}" = fg ]]; then
    # get the job id from the first argument passed to the fg command
    local job_id jobspec="${cmdargs[2]#%}"
    # logic based on jobs arguments:
    # http://zsh.sourceforge.net/Doc/Release/Jobs-_0026-Signals.html#Jobs
    # https://www.zsh.org/mla/users/2007/msg00704.html
    case "$jobspec" in
        <->) # %number argument:
            # use the same <number> passed as an argument
            job_id=${jobspec} ;;
        ""|%|+) # empty, %% or %+ argument:
            # use the current job, which appears with a + in $jobstates:
            # suspended:+:5071=suspended (tty output)
            job_id=${(k)jobstates[(r)*:+:*]} ;;
        -) # %- argument:
            # use the previous job, which appears with a - in $jobstates:
            # suspended::-6493=suspended (signal)
            job_id=${(k)jobstates[(r)*::-*]} ;;
        [?]* ) # %?string argument:
            # use $jobtexts to match for a job whose command *contains* <string>
            job_id=${(k)jobtexts[(r)*${(Q)jobspec}*]} ;;
        *) # %string argument:
            # use $jobtexts to match for a job whose command *starts with* <string>
            job_id=${(k)jobtexts[(r)${(Q)jobspec}*]} ;;
    esac

    # override preexec function arguments with job command
    if [[ -n "${jobtexts[$job_id]}" ]]; then
        1="${jobtexts[$job_id]}"
        2="${jobtexts[$job_id]}"
    fi
fi

# cmd name only, or if this is sudo or ssh, the next cmd
local CMD=${1[(wr)^(=*|sudo|ssh|mosh|rake|-*)]:gs/%/%%}
local LINE="${2:gs/%/%%}"

title '$CMD' '%100>...>$LINE%<<'
}

```

```
autoload -U add-zsh-hook
add-zsh-hook precmd mzc_termsupport_precmd
add-zsh-hook preexec mzc_termsupport_preexec
```

.nvmrc detection

```
# place this after nvm initialization!
autoload -U add-zsh-hook
load-nvmrc() {
  local node_version="$(nvm version)"
  local nvmrc_path="$(nvm_find_nvmrc)"

  if [ -n "$nvmrc_path" ]; then
    local nvmrc_node_version=$(nvm version "$(cat "${nvmrc_path}")")

    if [ "$nvmrc_node_version" = "N/A" ]; then
      nvm install
    elif [ "$nvmrc_node_version" != "$node_version" ]; then
      nvm use
    fi
  elif [ "$node_version" != "$(nvm version default)" ]; then
    echo "Reverting to nvm default version"
    nvm use default
  fi
}
add-zsh-hook chpwd load-nvmrc
load-nvmrc
```

npm completions

```
_zbnrc_npm_command() {
  echo "${words[2]}"
}
```

```

_zbnc_npm_command_arg() {
    echo "${words[3]}"
}

_zbnc_no_of_npm_args() {
    echo "$#words"
}

_zbnc_list_cached_modules() {
    ls ~/.npm 2>/dev/null
}

_zbnc_recursively_look_for() {
    local filename="$1"
    local dir=$PWD
    while [ ! -e "$dir/$filename" ]; do
        dir=${dir%/*}
        [[ "$dir" = "" ]] && break
    done
    [[ ! "$dir" = "" ]] && echo "$dir/$filename"
}

_zbnc_get_package_json_property_object() {
    local package_json="$1"
    local property="$2"
    cat "$package_json" |
        sed -nE "/^  \\"$property\\": \{$/,/^  \},?$/p" | # Grab scripts object
        sed '1d;$d' | # Remove first/last lines
        sed -E 's/    "([^\"]+)": "(.+)",?/\1=>\2/' # Parse into key=>value
}

_zbnc_get_package_json_property_object_keys() {
    local package_json="$1"
    local property="$2"
    _zbnc_get_package_json_property_object "$package_json" "$property" | cut -f 1 -d "="
}

_zbnc_parse_package_json_for_script_suggestions() {
    local package_json="$1"

```

```

_zbnc_get_package_json_property_object "$package_json" scripts |
  sed -E 's/(.+)=>(.)/\1:$ \2/' | # Parse commands into suggestions
  sed 's/\(:\)[^$]/\&/g' |      # Escape ":" in commands
  sed 's/\(:\)$[^ ]/\&/g'      # Escape ":" without a space in commands
}

_zbnc_parse_package_json_for_deps() {
  local package_json="$1"
  _zbnc_get_package_json_property_object_keys "$package_json" dependencies
  _zbnc_get_package_json_property_object_keys "$package_json" devDependencies
}

_zbnc_npm_install_completion() {

  # Only run on `npm install ?`
  [[ ! "$(_zbnc_no_of_npm_args)" = "3" ]] && return

  # Return if we don't have any cached modules
  [[ "$(_zbnc_list_cached_modules)" = "" ]] && return

  # If we do, recommend them
  _values "$(_zbnc_list_cached_modules)

  # Make sure we don't run default completion
  custom_completion=true
}

_zbnc_npm_uninstall_completion() {

  # Use default npm completion to recommend global modules
  [[ "$(_zbnc_npm_command_arg)" = "-g" ]] || [[ "$(_zbnc_npm_command_arg)" = "--global" ]] &&
return

  # Look for a package.json file
  local package_json="$(_zbnc_recursively_look_for package.json)"

  # Return if we can't find package.json
  [[ "$package_json" = "" ]] && return
}

```

```

_values $_zbncc_parse_package_json_for_deps "$package_json")

# Make sure we don't run default completion
custom_completion=true
}

_zbncc_npm_run_completion() {

# Only run on `npm run ?`
[[ ! "$_zbncc_no_of_npm_args" = "3" ]] && return

# Look for a package.json file
local package_json="$_zbncc_recursively_look_for package.json"

# Return if we can't find package.json
[[ "$package_json" = "" ]] && return

# Parse scripts in package.json
local -a options
options=($(f)"$_zbncc_parse_package_json_for_script_suggestions $package_json")

# Return if we can't parse it
[[ "$#options" = 0 ]] && return

# Load the completions
_describe 'values' options

# Make sure we don't run default completion
custom_completion=true
}

_zbncc_default_npm_completion() {
  compadd -- $(COMP_CWORD=$((CURRENT-1)) \
    COMP_LINE=$BUFFER \
    COMP_POINT=0 \
    npm completion -- "${words[@]}" \
    2>/dev/null)
}

```

```
_zbnc_zsh_better_npm_completion() {

    # Store custom completion status
    local custom_completion=false

    # Load custom completion commands
    case "$(_zbnc_npm_command)" in
        i|install)
            _zbnc_npm_install_completion
            ;;
        r|uninstall)
            _zbnc_npm_uninstall_completion
            ;;
        run)
            _zbnc_npm_run_completion
            ;;
    esac

    # Fall back to default completion if we haven't done a custom one
    [[ $custom_completion = false ]] && _zbnc_default_npm_completion
}

compdef _zbnc_zsh_better_npm_completion npm
```

Final `.zshrc`

```
source /usr/share/nvm/init-nvm.sh

source ~/.zsh/1_general.zsh
source ~/.zsh/2_keybindings.zsh
source ~/.zsh/3_aliases.zsh
source ~/.zsh/4_theming.zsh
source ~/.zsh/5_plugins.zsh
source ~/.zsh/6_termwin_func.zsh
source ~/.zsh/7_nvmsrc.zsh
source ~/.zsh/8_npm_completion.zsh
```

powerlevel10k prompt theme

```
yay -S zsh-theme-powerlevel10k-git nerd-fonts-jetbrains-mono  
echo 'source /usr/share/zsh-theme-powerlevel10k/powerlevel10k.zsh-theme' >> ~/.zshrc
```

Revision #10

Created 2021-04-02 21:48:53 UTC by Sebin

Updated 2023-03-26 03:28:21 UTC by Sebin