Customization & Tweaks

Make it yours

- DBus
- KDE Plasma Themes
- Additional Packages
- zsh configuration
- Plymouth
- Reinstall preparation
- Qt Wayland
- Removing unused packages (orphans)

DBus

`dbus-broker` is a drop-in replacement for the *libdbus* reference implementation, which aims "to provide high performance and reliability, while keeping compatibility to the D-Bus reference implementation".

Install `dbus-broker`:

```
pacman -S dbus-broker
```

Disable `dbus.service` and enable `dbus-broker.service` globally:

```
systemctl disable dbus
systemctl enable dbus-broker
```

Reboot for the changes to take effect.

KDE Plasma Themes

Akava

https://akava-design.github.io/

```
yay -S breeze-blurred-git akava-kde-git akava-konsole-git kvantum-theme-akava-git
```

Layan

```
yay -S kvantum-theme-layan-git layan-cursor-theme-git layan-gtk-theme-git layan-kde-git tela-icon-theme-git
```

Aritim

Install Konsole theme with "Get new stuff" Feature from Konsole.

```
yay -S aritim-dark-gtk-git aritim-light-kde-git aritim-dark-kde-git aritim-light-gtk-git lightly-qt kora-icon-theme
```

Additional Packages

Additional packages one might find useful to have installed on their system

File System Utilities

File system	Package	Creation command	Description
Btrfs	`btrfs-progs`	`mkfs.btrfs`	Btrfs filesystem utilities
VFAT	`dosfstools`	`mkfs.fat`	DOS filesystem utilities
exFAT	`exfatprogs`	`mkfs.exfat`	exFAT filesystem userspace utilities for the Linux Kernel exfat driver
F2FS	`f2fs-tools`	`mkfs.f2fs`	Tools for Flash-Friendly File System (F2FS)
ext4	`e2fsprogs`	`mkfs.ext4`	Ext2/3/4 filesystem utilities
HFS/HFS+	`hfsprogs` (AUR)	`mkfs.hfsplus`	User space utils to create and check Apple HFS/HFS+ filesystem
JFS	`jfsutils`	`mkfs.jfs`	JFS filesystem utilities
NILFS2	`nilfs-utils`	`mkfs.nilfs2`	A log-structured file system supporting continuous snapshotting
NTFS	`ntfs-3g`	`mkfs.ntfs`	NTFS filesystem driver and utilities
ReiserFS	`reiserfsprogs`	`mkfs.reiserfs`	Reiserfs utilities
UDF	`udftools`	`mkfs.udf`	Linux tools for UDF filesystems and DVD/CD-R(W) drives
XFS	`xfsprogs`	`mkfs.xfs`	XFS filesystem utilities

System Tools

Package	Description

`fwupd`	Simple daemon to allow session software to update firmware
`htop`	Interactive process viewer
`impression`	A straight-forward modern application to create bootable drives.
`libimobiledevice`	Library to communicate with services on iOS devices using native protocols
`lshw`	A small tool to provide detailed information on the hardware configuration of the machine
`man-db`	A utility for reading man pages
`power-profiles-daemon`	Makes power profiles handling available over D-Bus
`radeontop`	View AMD GPU utilization for total activity percent and individual blocks

Multimedia

Package	Description
`aegisub`	A general-purpose subtitle editor with ASS/SSA support
`amberol`	Plays music, and nothing else
`blanket`	Improve focus and increase your productivity by listening to different sounds
`eartag`	Simple music tag editor
`handbrake`	Multithreaded video transcoder
`identity`	Compare multiple versions of an image or video
`makemkv`	DVD and Blu-ray to MKV converter
`mediainfo-gui`	Supplies technical and tag information about media files
`mousai`	Simple application for identifying songs
`paleta`	Extract the dominant colors from any image
`soundconverter`	A simple sound converter application for GNOME
`tube-converter`	An easy-to-use video downloader
`video-trimmer`	Trim videos quickly

Productivity

Package	Description
`apostrophe`	A distraction free Markdown editor for GNU/Linux made with GTK+

Developer Tools

Package	Description
`devtoolbox`	Development tools at your fingertips
`escambo`	An HTTP-based APIs test application for GNOME
`gitg`	GNOME GUI client to view git repositories
`ohmysvg`	SVG optimizer
`playhouse`	A Playground for HTML/CSS/JavaScript
`share-preview`	Preview and debug websites metadata tags for social media share
`textpieces`	Transform text without using random websites
`visual-studio-code-bin`	Editor for building and debugging modern web and cloud applications
`webfontkitgenerator`	Create `@font-face` kits easily

Misc Tools

Package	Description
`extension-manager`	A native tool for browsing, installing, and managing GNOME Shell Extensions
`gnome-obfuscate`	Censor private information
`newsflash`	Desktop application designed to complement an already existing web-based RSS reader account
`p7zip`	Command-line file archiver with high compression ratio
`teamviewer`	All-In-One Software for Remote Support and Online Meetings

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 expension 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
HISTSIZE=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 '^[[3;5~' kill-word                                        # delete next word with
ctrl+del
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

```
_zbnc_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 $(_zbnc_parse_package_json_for_deps "$package_json")

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

_zbnc_npm_run_completion() {

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

  # Parse scripts in package.json
  local -a options
  options=(${(f)"$(_zbnc_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
}

_zbnc_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
```

Command not found handler

```
function command_not_found_handler {
    local purple='\e[1;35m' bright='\e[0;1m' green='\e[1;32m' reset='\e[0m'
    printf 'zsh: command not found: %s\n' "$1"
    local entries=(
        ${(f)"$(/usr/bin/pacman -F --machinereadable -- "/usr/bin/$1")"}
    )
    if (( ${#entries[@]} ))
```

```
    then
        printf "${bright}$1${reset} may be found in the following packages:\n"
        local pkg
        for entry in "${entries[@]}"
        do
            # (repo package version file)
            local fields=(
                ${(0)entry}
            )
            if [[ "$pkg" != "${fields[2]}" ]]
            then
                printf "${purple}%s/${bright}%s ${green}%s${reset}\n" "${fields[1]}"
"${fields[2]}" "${fields[3]}"
            fi
            printf '    /%s\n' "${fields[4]}"
            pkg="${fields[2]}"
        done
    fi
    return 127
}
```

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_nvmrc.zsh
source ~/.zsh/8_npm_completion.zsh
source ~/.zsh/9_command_not_found.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
```

Plymouth

Plymouth replaces boot messages with a pretty splash screen.

Installation

```
yay -S plymouth ttf-dejavu
```

Configuration

Enabling Plymouth requires editing the `HOOKS` array in `/etc/mkinitcpio.conf`. Depending on what your initramfs is based on the hooks slightly differ.

Busybox

If your initramfs is busybox-based (default in Arch Linux), add the `plymouth` hook **after** the `base` and `udev` hooks:

> **ATTENTION:** When using the `encrypt` hook to unlock encrypted devices during boot, place it **after** the `plymouth` hook in order to receive a passphrase prompt, e.g.:
>
> ```
> HOOKS=(base udev plymouth autodetect keyboard keymap consolefont modconf block encrypt lvm2 filesystems fsck)
> ```

```
HOOKS=(base udev plymouth ...)
```

Systemd

If your initramfs is systemd-based (i.e. to make use of `systemd-cryptenroll`), add the `plymouth` hook **after** the `base` and `systemd` hooks:

ATTENTION: When using the `sd-encrypt` hook to unlock encrypted devices during boot, place it **after** the `plymouth` hook in order to receive a passphrase prompt, e.g.:

```
HOOKS=(base systemd plymouth autodetect keyboard sd-vconsole modconf block sd-encrypt lvm2 filesystems fsck)
```

```
HOOKS=(base systemd plymouth ...)
```

Theming

A great selection of Plymouth themes can be found on the AUR.

To list available Plymouth themes (alternatively `ls /usr/share/plymouth/themes`):

```
plymouth-set-default-theme -l
```

Set the Plymouth theme and rebuild (`-R`) the initramfs, e.g. BGRT (keeps firmware logo and displays a spinner in a similar fashion to Windows):

TIP: When unlocking a LUKS encrypted root file system during boot the passphrase prompt replaces the firmware logo. To prevent this install and set the following theme instead:

```
yay -S plymouth-theme-bgrt-better-luks
sudo plymouth-set-default-theme -R bgrt-better-luks
```

```
sudo plymouth-set-default-theme -R bgrt
```

Reboot and enjoy!

Reinstall preparation

Backup

Folders in `/home`

- `.dosbox` (DOSBox configs)
- `.local/bin` (local scripts)
- `.mozilla` (Firefox profile)
- `.ssh` (SSH keys and configs)
- `DOS` (DOSBox root)
- `DOSGAMES` (ISO images)
- Downloads
- Run Sync NAS script to save Documents, Pictures, Music and Video
- Possible Windows game saves beneath `.wine` prefix

Configs

Folding@Home

Folding@Home config: `/etc/foldingathome/config.xml`

HandBrake

Export Configs via GUI

VS Code

Use Cloud Sync extension with GitHub

Qt Wayland

Display server

To utilize Wayland for Qt-based applications install the `qt5-wayland` and `qt6-wayland` packages. Optionally, also install `qt5ct` and `qt6ct` if you're on a non-KDE desktop environment.

Then set the `QT_QPA_PLATFORM` environment variable to:

- `wayland` for the wayland plugin
- `xcb` for the X11 plugin
- `qt6ct` for running Qt6-based applications on non-KDE desktop environments
- `qt5ct` for running Qt5-based applications on non-KDE desktop environments

> **TIP:** It may prove useful to set multiple values separated by `;`. In case one is not available, the next one is used.

```
QT_QPA_PLATFORM="wayland;qt5ct;xcb"
```

Use KDE dialogs

If some applications (e.g. Telegram) don't use default KDE dialogs, set the following environment variable:

```
QT_QPA_PLATFORMTHEME="flatpak"
```

Removing unused packages (orphans)

Orphans can accumulate as packages are removed via `pacman -R` instead of `pacman -Rs`, makedepends or packages removing dependencies in subsequent versions. These can accumulate over time and waste space

To recursively remove packages that are not required by other packages (including their configuration files) installed on the systems use the following command:

Parameter	Description
`-Q`	Query the local database
`-t`	List packages not required by any other installed package (`-tt` to also list packages installed as optional dependencies)
`-d`	List packages installed as dependencies
`-q`	Show less information (e.g. only package names, useful for piping)
`-R`	Remove packages
`-n`	Also remove configuration files
`-s`	Remove unneeded packages recursively

```
pacman -Qtdq | pacman -Rns -
```