If you don't find what you're looking for, and you think it should be covered by the FAQ, please open a new issue with what you think should be here.

Definitions

What is Oh My Zsh and what does it have to do with zsh?

Zsh is a shell, just like bash or fish, which interprets commands and runs them. Oh My Zsh is a framework built on top of zsh that is structured to allow it to have plugins and themes, as well as providing what we think are the best settings from the start. You can use zsh without Oh My Zsh, but you can't use Oh My Zsh if you don't have zsh.

What is a zsh plugin?

A zsh plugin is, on its most basic form, just a shell script that can be interpreted by Zsh. So, in its essence, it can do anything that you could do run line by line in your terminal. A plugin can also provide completion for certain commands and functions.

What is a zsh theme?

A zsh theme is a zsh script that changes the **prompt**. The prompt is the line (or lines, on multiline themes) that appear every time you want to run a command. This is ultimately accomplished by changing the `$PROMPT` variable.

But a theme can do much more than that, like adding an automatic git status prompt that changes each time the prompt is redrawn, or a clock that is refreshed per-second, or many other functionalities like that. Browse the [[Themes]] page to find one that you like.

NOTE: a zsh theme can't change the appearance of your terminal emulator. This includes things like the color scheme or the character font. Those are separate settings that need to be configured in your terminal emulator settings.

What is the .zshrc file?

The `.zshrc` file (or just `zshrc` file) is a file that zsh reads when it starts. It can be found in your home directory, which will change depending on the system and platform you're using. On a terminal, the path to the home directory can be found by running `echo $HOME`.

What Oh My Zsh is not

- Zsh
- A Zsh replacement
- A plugin/theme downloader (although a lot of plugins/themes are preinstalled)
- The app that's called "Terminal"
- A terminal shell
- A Zsh theme
- A Bash theme
- A dead project

What Zsh is not

- Oh My Zsh
- A plugin engine
- A reason to instantly uninstall Bash
- Unusable without Oh My Zsh
- The app that's called "Terminal"

How do I...?

How do I install Zsh?

Check first that you have it installed: run `zsh`. If you really don't have it installed, please see [[Installing Zsh]] for instructions on how to install it on multiple systems.

How do I install Zsh on Windows?

You can't install Zsh directly on Windows. As a reminder, Oh My Zsh needs Zsh installed to work - if you can't install Zsh, you can't install Oh My Zsh.

To use Zsh on Windows, you need Windows 10 2004 or 11, and one of the following:

- Cygwin
- WSL (note that this requires you to first install WSL)

If you're running earlier versions of Windows, you can't install it at all. You'll need a virtual machine, or a proper Linux install.

How do I install Oh My Zsh?

Please follow the project's README instructions for a basic installation, or the advanced instructions if you need to automate the installation or change some of the settings of the installer.

How do I uninstall Oh My Zsh?

To remove Oh My Zsh, run `uninstall_oh_my_zsh`. The uninstaller will:

1. Delete the Oh My Zsh folder.
2. Back up your current `.zshrc` file to something like `.zshrc.omz-uninstalled-<date>`. This is so you can recover any settings you might have written in there. If you haven't, you can safely delete it afterwards.
3. If you had a `.zshrc` file previously to installing Oh My Zsh, the uninstaller will restore it back as the new `.zshrc` file.
4. It will attempt to switch back to the default shell you had before installing Oh My Zsh. This will be done by reading the file `~/.shell.pre-oh-my-zsh` which should have been created by the installer; otherwise, it will use `/bin/bash` as the new default shell.

Alternatively, to manually remove OMZ do `rm -rf $ZSH` — you will also have to edit your `.zshrc` file. To manually set a new default shell, do `chsh -s /my/new/shell`, *e.g.* `chsh -s /bin/bash`. Then simply reopen your shells.

How do I change my locale?

The `locale` is used in a shell environment to define which language and character encoding (i.e. UTF-8) to use. This will change the language used by commands, as well as the encoding of the characters. Usually, you'll have set the language to use in the settings of your particular system. But sometimes you need to change the encoding used in the shell, most frequently to set it to use UTF-8.

First, to verify what locale you're using, run `locale`. You'll get something like this:

```
$ locale
LANG=en_US.utf8
LANGUAGE=
LC_CTYPE=en_US.utf8
LC_NUMERIC="en_US.utf8"
LC_TIME="en_US.utf8"
LC_COLLATE="en_US.utf8"
LC_MONETARY="en_US.utf8"
LC_MESSAGES="en_US.utf8"
LC_PAPER="en_US.utf8"
LC_NAME="en_US.utf8"
LC_ADDRESS="en_US.utf8"
LC_TELEPHONE="en_US.utf8"
LC_MEASUREMENT="en_US.utf8"
LC_IDENTIFICATION="en_US.utf8"
LC_ALL=
```

These are environment variables that each change one particular part of the messages that commands print. For example, `LC_COLLATE` changes the order in which strings are sorted. But you don't need to know them. We're only interested in `LANG` and `LC_ALL`.

`LC_ALL` defines the locale for all the rest of the locale environment variables. If it's not set, then `LANG` takes precedence. In the case above, it is set to use the `English (United States)` language, and a `UTF-8` encoding.

If we need to change these, we need to see what available locales are installed, and we do that by running `locale -a` (locales ending in `.UTF-8` or `.utf8` all use UTF-8 as their encoding):

```
$ locale -a
C
C.UTF-8
en_US.utf8
POSIX
```

Once we have selected a suitable locale (tip: always use a UTF-8 locale), we can set it in our zshrc file, preferably at the top:

```
export LANG=en_US.utf8
```

How do I reload the zshrc file?

You may have seen somewhere that when you make a change to your zshrc file, you need to reload it. The common **wrong** suggestion is to use "source ~/.zshrc". This may cause trouble because some things that are already in the zsh session haven't been removed (variables, functions, hooks...). It's also possible that you're repeatedly running an init script and causing more and more processes to start.

To properly reload the zshrc file, you need to restart the zsh session. You can either:

- Restart the terminal.

or

- Restart the zsh process by running: `exec zsh`.

How do I reset the completion cache?

The completion cache file (also known as zcompdump file) is a cache of the completion functions found when starting the zsh session. This file is written when calling `compinit`, and it is done automatically by Oh My Zsh when starting. It can be found at `$ZSH_COMPDUMP`, usually in your home directory and named `.zcompdump-<host>-<zsh-version>`.

If there is already a file at that location, `compinit` will read it, instead of recreating it, for better performance. This causes problems sometimes when some completion settings where changed but the completion cache wasn't, like when a plugin is enabled. That's why most completion problems are solved by resetting the cache and restarting the zsh session.

To reset it, you have to delete it and restart your zsh session so that it is recreated again:

```
# Delete the completion cache
rm "$ZSH_COMPDUMP"
# Restart the zsh session
exec zsh
```

How do I manually update Oh My Zsh from a script?

The best way to do that is to call the `upgrade.sh` script directly. If `$ZSH` is defined (it should point to where OMZ is installed), then you can call it like so:

```
"$ZSH/tools/upgrade.sh"
```

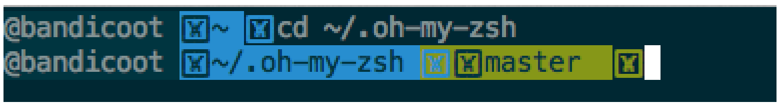
Another option is to use the `omz` CLI. The `omz update` command triggers an update, but it also shows the changelog and restarts the zsh session (which *might* cause a restart loop). To avoid that, run it with the `--unattended` flag, like so: `omz update --unattended`. But again, it's better to call the `upgrade.sh` script directly.

Common problems

Font issues

I have a weird character in my prompt

It might look like this in iTerm2:



```
@bandicoot ⊠~ ⊠cd ~/.oh-my-zsh
@bandicoot ⊠~/.oh-my-zsh ⊠⊠master ⊠
```

Some themes like Agnoster require a special font to render certain characters. First, make sure you have read the theme's instructions written in the [[Themes]] section. Most of the time this issue is solved by installing a Powerline font, or a nerd font. Some themes may require the use of a nerd font over a powerline font, but nerd fonts do work whereever powerline fonts work.

After installing it, you need to enable the font in the settings of your terminal emulator. Just installing the font isn't enough, as the terminal doesn't know to use that font for the symbols. Also note that the font has to be set anywhere you display the themes - this may include editors and IDEs.

Completion issues

Most completion issues are due to an old completion cache file (also called zcompdump file). Before attempting anything else, try to reset it following the instructions above. If that doesn't make the completion work, look at the other completion issue sections.

I have enabled a completion plugin but the completion doesn't work

> ***NOTE: a recent update checks whether the plugin list has changed (more accurately, `$fpath`) and resets the zcompdump file, so the solution below isn't needed anymore. If you don't have that yet, update Oh My Zsh with `omz update`.***

As of right now, after enabling or disabling a plugin that provides completion, you have to reset the cached completion file. Follow the instructions above to delete the completion file.

If this doesn't solve the problem, you might have found a bug in the plugin. Search for issues with the plugin in question, and if there isn't one already (open or closed), open a new bug report.

I see duplicate typed characters after I complete a command

This usually happens because your theme uses UTF-8 characters but your `locale` is not set up to handle them. Make sure that you're running a `locale` ending in `.UTF-8` or `.utf8`. See How do I change my locale for instructions.

Similar issues: #6985, #3932, #4529, #4632.

Zsh errors

zsh: no matches found

This error happens when you used a wildcard character (also called glob operators or globbing characters), which indicate to the zsh interpreter to look for files matching that wildcard pattern (see Filename generation for the full documentation). The most common examples of these are `*` (star) and `?` (question mark), when used in anything from `apt`, `git` or even `curl` commands (`?` is usually part of a URL).

There are many solutions, some temporary, some permanent:

1. Temporary: wrap the arguments containing wildcards in **quotes** (double quotes or single quotes, both are OK). For example:

```
$ apt install linux-*
zsh: no matches found: linux-*
$ apt install 'linux-*'
# the command continues successfully
```

2. Temporary: prepend **`noglob`** to the command, such that any wildcards will be ignored.

```
$ noglob apt install linux-*
# has the same effect as
$ apt install 'linux-*'
```

3. Permanent: disable globbing (*aka* wildcard expansion) at all, using **`unsetopt glob`**. Put it somewhere in your zshrc file **after** Oh My Zsh is sourced so that it's applied on every zsh session. NOTE: this will mean that you won't be able to use wildcards anywhere in your zsh session.

```
$ unsetopt glob
$ ls -d .* #
ls: cannot access '.*': No such file or directory
```

4. Permanent: another option is using the solution in (2) but making it stick with **an alias** so that you don't need to do this every time you run that particular command. This is much more fine-grained than (3) because you can still use wildcards in other commands.

```
$ alias apt='noglob apt'
$ apt install linux-*
# the command continues successfully
```

Other problems

`kill-word` or `backward-kill-word` do / don't delete a symbol (`WORDCHARS`)

History:

Since the beginning of Oh My Zsh and up until commit 50dc4ab (Sept. 4, 2020), the framework set `WORDCHARS` to empty string. Afterwards, **`WORDCHARS` was set to `_-`**. Days after that, there was ample support for going back to the previous behavior, so commit 3f42700c reverted the change.

What does this do?

This variable tells zsh which non-alphanumeric characters are part of a **word**. This means that any characters in this string will be included in what constitutes a word. If `WORDCHARS` is `''`, that means that only alphanumeric characters are part of a word. Let these examples explain it better (the `|` represents the cursor):

```
$ command arg1 arg2-with_symbols|
```

Let's imagine we press `CTRL-Backspace` which, as of recently, runs `backward-kill-word`, so it deletes a word to the left of the cursor. If `WORDCHARS` is empty string (`''`), only the alphanumeric characters are interpreted to be part of the word. So only characters up until the last underscore will be deleted:

```
$ command arg1 arg2-with_|
```

If instead, as Oh My Zsh does now, `WORDCHARS='_-'`, hyphens and underscores are also part of a word. So when we press `CTRL-Backspace`, the whole arg2 will be deleted, since all of it constitutes a word:

```
$ command arg1 |
```

This becomes more helpful *when trying to delete dash arguments*. For instance, after deleting the word to the left, this:

```
$ git commit --all --dry-run|
```

becomes this:

```
$ git commit --all |
```

The default Zsh value of `WORDCHARS` is, as of version 5.7.1:

```
WORDCHARS='*?_-.[]~=/&;!#$%^(){}<>'
```

If you want this behavior to change, set the `WORDCHARS` variable in your zshrc file, **after Oh My Zsh is sourced**. So if you'd wanted a star (`*`) to also be a part of a word, as well as the old ones (hyphen and underscore), you'd set the following:

```
WORDCHARS='_-*'
```