News

▶ Here are the new features and updates added to Zinit in the last 90 days.

- 08-11-2021
  - Zinit Packages for `zinit pack ...` listed in #zinit-package topic.
  - Zinit Annexes (i.e. an extension) listed in #zinit-annex topic.
  - Zinit Services listed in #zinit-service topic.
  - Zinit Wiki mostly restored.
- 05-11-2021
  - The packages have been disconnected from NPM registry and now live only on z-shell organization under #zinit-package topic. Publishing to NPM isn't needed.
  - There are two interesting packages, any-gem and any-node. They allow to install any Gem(s) or Node module(s) locally in a newly created plugin directory. For example:

    ```
    zinit pack param='GEM -> rails' for any-gem
    zinit pack param='MOD -> doctoc' for any-node
    # To have the command in zshrc, add an id-as'' ice so that
    # Zinit knows that the package is already installed
    # (also: the Unicode arrow is allowed)
    zinit id-as=jekyll pack param='GEM → jekyll' for any-gem
    ```

    The binaries will be exposed without altering the PATH via shims (Bin-Gem-Node annex is needed). Shims are correctly removed when deleting a plugin with `zinit delete …`.
- 16-07-2020
  - A new ice `null` which works exactly the same as `as"null"`, i.e.: it makes the plugin a *null*-one ↔ without any scripts sourced (by default, unless `src''` or `multisrc''` are given) and compiled, and without any completions searched / installed. Example use case:

```
zi null sbin"vims" for MilesCranmer/vim-stream
```

instead of:

```
zi as"null" sbin"vims" for MilesCranmer/vim-stream
```

.

- A **new annex Unscope** 😃 It's goal is: to allow the usage of the unscoped — i.e.: given without any GitHub user name — plugin IDs. Basically it allows to specify, e.g.: **zinit load *zsh-syntax-highlighting*** instead of **zinit load *zsh-users/zsh-syntax-highlighting***. It'll automatically send a request to the GitHub API searching for the best candidate (max. # of stars and of forks). It also has an embedded, static database of short *nicknames* for some of the plugins out there (requests for addition are welcomed), e.g.: **vi-reg** for **zsh-vi-more/evil-registers**.
- A fresh and elastic hook-based architecture has been implemented and deployed — the code is much cleaner and the development will be easier, i.e.: quicker 😃.
- Set of small improvements: **a)** `silent''` mutes the `Snippet not loaded` error message, **b)** much shorter lag/pause after a plugin installation or update, **c)** the 256 color palette is being now used for plugin IDs, if available, **d)** if possible (a UTF-8 locale is needed to be set), the Unicode three-dots … will be used instead of `...` in the messages, **e)** nicer snippet IDs in the installation and update messages, **f)** the annexes can be now loaded in any order without influencing their operation in any way (there have been some issues with Patch-Dl and As-Monitor annexes), **g)** `compile''` can now obtain multiple patterns separated via semicolon (`;`).

- 25-06-2020
  - Ability to call the autoloaded function at the moment of loading it by `autoload'#fun'`, i.e.: by prefixing it with the hash sigh (`#`). So that it's possible to invoke e.g.:

```
zinit autoload'#manydots-magic' for knu/zsh-manydots-magic
```

instead of:

```
zinit autoload'manydots-magic' atload'manydots-magic' for \
    knu/zsh-manydots-magic
```

- 20-06-2020

- The Bin-Gem-Node annex now has an explicit Cygwin support — it creates additional, **extra shim files** — Windows batch scripts that allow to run the shielded applications from e.g.: Windows run dialog — if the `~/.zinit/polaris/bin` directory is being added to the Windows `PATH` environment variable, for example (it is a good idea to do so, IMHO). The Windows shims (*shims* are command-wrapper scripts that are in general created with the `sbin''` ice of the annex) have the same name as the standard ones (which are also being created, normally) plus the `.cmd` extension. You can test the feature by e.g.: installing Firefox from the Zinit package via:

```
zinit pack=bgn for firefox
```

- All cURL progress bars are now guaranteed to be single line — this is being done by a wrapper script.

- I thought that I'll share an interesting function-type that I'm using within Zinit - a function that outputs messages with theming and colors easily available:

```
typeset -gA COLORS=(
    col-error  $'\e[31m'
    col-file   $'\e[38;5;110m'
    col-url    $'\e[38;5;45m'
    col-meta   $'\e[38;5;221m'
    col-meta2  $'\e[38;5;154m'
    col-data   $'\e[38;5;82m'
    col-data2  $'\e[38;5;50m'
    col-rst    $'\e[0m'
    col-can-be-empty ""
)

m() {
    builtin emulate -LR zsh -o extendedglob
    if [[ $1 = -* ]] { local opt=$1; shift } else { local opt }
    local msg=${(j: :)${@//(#b)([\[\{]([^\]\}]##)
[\]\}])/${COLORS[col-$match[2]]-$match[1]}}}
    builtin print -Pr ${opt:#--} -- $msg
}
```

Usage is as follows:

```
m "{error}ERROR:{rst} The {meta}data{rst} has the value:
{data}value{rst}"
```

Effect:

The function is available in the `atinit''`, `atload''`, etc. hooks.

- 17-06-2020
  - `ziextract` and `extract''` now support Windows installers — currently the installer of Firefox. Let me know if any of your installers doesn't work. You can test the installer with the Firefox Developer Edition Zinit package:

    ```
    zinit pack"bgn" for firefox-dev
    ```

    The above command will work on Windows (at least on Cygwin), Linux and OS X.

- 13-06-2020
  - `ziextract` has a new `--move2` option, which moves files two levels up after unpacking. For example, if there will be an archive file with directory structure: `Pulumi/bin/{pulumi,pulumi2}`, then after `ziextract --move2 --auto` there will be the two files moved to the top level dir: `./{pulumi,pulumi2}`. To obtain the same effect using the `extract''` ice, pass two exclamation marks, i.e.: `extract'!!'`. A real-world example — it uses z-a-as-monitor and z-a-bin-gem-node annexes to download a Zip package that has the files inside two-level nested directory tree:

    ```
    zi id-as`pulumi` as`monitor|null` mv`pulumi pulumi_` extract`!` \
        dlink=`https://get.pulumi.com/releases/sdk/pulumi-%VERSION%-
    windows-x64.zip` \
        sbin`pulumi*` for \
            https://www.pulumi.com/docs/get-started/install/versions/
    ```

- 12-06-2020
  - New options to `update`: `-s/--snippets` and `-l/--plugins` — they're limiting the `update --all` to only plugins or snippets. Example:

    ```
    zinit update --plugins
    ```

    Work also with `-p/--parallel`.

- 15-05-2020
  - The `autoload''` ice can now rename the autoloaded functions, i.e.: load a function from a file `func-A` as a function `func-B` via: `autoload'func-A -> func-B; …'`.

  - Also, an alternate autoloading method - via: `eval "func-file() { $(<func-file); }"` — has been exposed — in order to use it, precede the ice contents with an exclamation mark, i.e.: `autoload'!func-file'`. The rename mode uses this method by default.

- 12-05-2020

  - A new feature — ability to substitute `stringA` → `stringB` in plugin source body before executing by `subst'A -> B'`. Works also for any nested `source` commands. Example — renaming the `dl''` ice into a `dload''` ice in the Patch-Dl annex:

```
zinit subst"dl'' -> dload''" for z-shell/z-a-patch-dl
```

  - A new ice `autoload''` which invokes `autoload -Uz …` on the given files/functions. Example — a plugin that converts `cd ...` into `cd ../..` that lacks proper setup in any `*.plugin.zsh` file:

```
zinit as=null autoload=manydots-magic atload=manydots-magic for \
    knu/zsh-manydots-magic
```

- 09-05-2020

  - The `from'gh-r'` downloading of the binary files from GitHub releases can now download **multiple files** — if you specify multiple `bpick''` ices **or** separate the patterns with a semicolon (`;`). Example:

```
zinit from"gh-r" as"program" mv"krew-* -> krew" bpick"*.yaml"
bpick"*.tar.gz" for \
kubernetes-sigs/krew
```

  - A new ice `opts''` which takes options to **sticky-set** during sourcing of the plugin. This means that thee options will be also set for all of the *functions* that the plugin defines — **during their execution** (***only***). The option list is space separated. Example:

```
# Suppose the example test plugin has the following in
test.plugin.zsh:
#
# print $options[kshglob] $options[shglob]
#
# Then:

zinit opts"kshglob noshglob" for z-shell/test

# Outputs:
on off
```

```
# Can mix with the standard emulation-ices: sh, bash, ksh, csh, zsh
(the
# default one)

zinit sh opts"kshglob" for z-shell/test

# Outputs `on' for the SH_GLOB, because sh-emulation sets this option
on on
```

- 07-05-2020
  - A new `from''` value is available — `cygwin`. It'll cause to download a package from the Cygwin repository — from a random mirror, and then unpack it. Example use:

```
# Install gzip and expose it through Bin-Gem-Node annex's sbin'' ice
zinit from"cygwin" sbin"usr/bin/gzip.exe -> gzip" for gzip
```

- 16-04-2020
  - Turbo plugins will now get gracefully preinstalled first before the prompt (i.e.: within `zshrc` processing) and then loaded **still** as Turbo plugins.
- 15-04-2020
  - The `…/name.plugin.zsh` and `…/init.zsh` can be now skipped from single-file (non-svn) snippet URLs utilizing the `OMZ::…`, etc. shorthands. Example:

```
# Instead of: zinit for OMZP::ruby/ruby.plugin.zsh
zinit for OMZP::ruby
# Instead of: zinit for PZTM::rails/init.zsh
zinit for PZTM::rails
# Instead of: zinit for OMZT::gnzh.zsh-theme
zinit for OMZT::gnzh
```

  - New prefixes `OMZP::` = `OMZ::/plugins/`, `OMZT::` = `OMZ::/themes/`, `OMZL::` = `OMZ::lib/`, `PZTM::` = `PZT::modules/`, for both svn and single-file snippets. Example use:

```
zinit for OMZP::ruby/ruby.plugin.zsh
zinit svn for OMZP::ruby
```

    (instead of:

```
zinit for OMZ::plugins/ruby/ruby.plugin.zsh
zinit svn for OMZ::plugins/ruby
```

).

- 12-04-2020
  - A new document on the Wiki is available — about the **bindmap" ice**.
  - If `id-as''` will have no value, then it'll work as **id-as'auto'**.
- 07-04-2020
  - A new feature — `param''` ice that defines params for the time of loading of the plugin or snippet. E.g.:

```
# Equivalent of `local myparam=1 myparam2=1' right before loading of
the plugin
zinit param'myparam → 1; myparam2 -> 1' for z-shell/null
# Equivalent of `local myparam myparam2' before loading of the plugin
zinit param'myparam; myparam2' for z-shell/null
```

  - The `atinit''` ice can now be investigated — if it'll be prepended with `!`, i.e.: `atinit'!…'`.
- 01-04-2020
  - As a user noticed, Subversion isn't distributed with Xcode Command Line Tools anymore. Here's a helpful snippet that installs Subversion with use of Zinit.
- 27-02-2020
  - An **important fix** has been pushed — due to a bug Turbo has been disabled for non-for syntax invocations of Zinit. Issue `zinit self-update` to resolve the mistake.
    - If you haven't updated yesterday, please restrain from running `zinit update` immediately after `self-update`. Support for reloading Zinit after `self-update` has been pushed yesterday and after pulling this feature, you'll be able to freely invoke `self-update` and `update`.
- 26-02-2020
  - From now on `zinit self-update` reloads Zinit for the current session (after updating the plugin manager), and `zinit update --all/-p/--parallel` detects that `self-update` has been run in **another session** and also reloads Zinit right before performing the update. This way the update code is always the newest and consistent.
- 26-02-2020
  - If the loaded object (plugin or snippet) is not already installed when loading, then Turbo gets automatically disabled for this single loading of the object — it'll be installed before prompt, not after it and also immediately (without waiting the number of seconds given to `wait''`), i.e.: during the normal processing of `zshrc`, which intuitively is the expected behavior.
  - The additional disk accesses for the checks cost about 10 ms out of 150 ms (i.e.: the Zsh startup time increases from 140 ms to 150 ms). If you want, you may disable the feature by setting `$ZINIT[OPTIMIZE_OUT_DISK_ACCESSES]` to `1`.

    - A bug in Turbo has been fixed that was delaying the objects' loadings, especially when there were no keystrokes issued.

- 20-02-2020

    - A new feature - **parallel updates** of all plugins and snippets — Zinit runs series of spawned concurrent-job groups of size 15 to speed up the update process. To activate, pass `-p`/`--parallel` to `update`, e.g.:

      ```
      zinit update -p
      zinit update --parallel
      # Increase the number of jobs in a concurrent-set to 40
      zinit update --parallel 40
      ```

      See demos: asciicast1, asciicast2.

    - A new article is available on the Wiki — about the **`extract`** ice.

- 09-02-2020

  Note that the ice `extract` can handle files with spaces — to encode such a name use the non-breaking space (Right Alt + Space) in place of the in-filename spaces 😃.

- 07-02-2020

    - A new ice `extract` which extracts:
        - all files with recognized archive extensions like `zip`, `tar.gz`, etc.,
        - if no such files will be found, then: all files with recognized archive **types** (examined with the `file` command),
        - OR, IF GIVEN — the given files, e.g.: `extract'file1.zip file2.tgz'`,
        - the automatic searching for archives ignores files in sub-sub-directories and located deeper,
    - It has a `!` flag — i.e.: `extract'!…'` — it'll cause the files to be moved one directory-level up upon unpacking,
    - and also a `-` flag — i.e.: `extract'-…'` — it'll prevent removal of the archive after unpacking; useful to allow comparing timestamps with the server in case of snippet-downloaded file,
    - the flags can be combined, e.g.: `extract'!-'`,
    - also, the function `ziextract` has a new option `--auto`, which causes the automatic behavior identical to the empty `extract` ice.

- 21-01-2020

    - A few tips for the project rename following the field reports (the issues created by users):
        - the `ZPLGM` hash is now `ZINIT`,
        - the annexes are moved under z-shell organization.

- 19-01-2020

    - The name has been changed to **Zinit** based on the results of the poll.

    - In general, you don't have to do anything after the name change.
    - Only a run of `zinit update --all` might be necessary.
    - You might also want to rename your `zplugin` calls in `zshrc` to `zinit`.
    - Zinit will reuse `~/.zplugin` directory if it exists, otherwise it'll create `~/.zinit`.

- 15-01-2020

    - There's a new function, `ziextract`, which unpacks the given file. It supports many formats (notably also `dmg` images) — if there's a format that's unsupported please don't hesitate to make a request for it to be added. A few facts:
        - the function is available only at the time of the plugin/snippet installation,
        - it's to be used within `atclone` and `atpull` ices,
        - it has an optional `--move` option which moves all the files from a subdirectory up one level,
        - one other option `--norm` prevents the archive from being deleted upon unpacking.
    - snippets now aren't re-downloaded unless they're newer on the HTTP server; use this with the `--norm` option of `ziextract` to prevent unnecessary updates; for example, the firefox-dev package uses this option for this purpose,
    - GitHub doesn't report proper `Last-Modified` HTTP server for the files in the repositories so the feature doesn't yet work with such files.

- 11-12-2019

    - Zinit now supports installing special-Zsh NPM packages! Bye-bye the long and complex ice-lists! Check out the Wiki for an introductory document on the feature.

- 25-11-2019

    - A new subcommand `run` that executes a command in the given plugin's directory. It has an `-l` option that will reuse the previously provided plugin. So that it's possible to do:

```
zplg run my/plugin ls
zplg run -l cat \*.plugin.zsh
zplg run -l pwd
```

- 07-11-2019

    - Added a prefix-char: `@` that can be used before plugins if their name collides with one of the ice-names. For example `sharkdp/fd` collides with the `sh` ice (which causes the plugin to be loaded with the POSIX `sh` emulation applied). To load it, do e.g.:

```
zinit as"null" wait"2" lucid from"gh-r" for \
    mv"exa* -> exa" sbin"exa"  ogham/exa \
    mv"fd* -> fd" sbin"fd/fd"  @sharkdp/fd \
    sbin"fzf" junegunn/fzf-bin
```

i.e.: precede the plugin name with `@`. Note: `sbin''` is an ice added by the z-a-bin-gem-node annex, it provides the command to the command line without altering `$PATH`.

See the Zinit Wiki for more information on the for-syntax.

- 06-11-2019
  - A new syntax, called for-syntax. Example:

```
zinit as"program" atload'print Hi!' for \
    atinit'print First!' z-shell/null \
    atinit'print Second!' svn OMZ::plugins/git
```

The output:

```
First!
Hi!
Second!
Hi!
```

And also:

```
% print -rl $path | egrep -i '(/git|null)'
/root/.zinit/snippets/OMZ::plugins/git
/root/.zinit/plugins/z-shell---null
```

To load in light mode, use a new `light-mode` ice. More examples and information can be found on the Zinit Wiki.

- 03-11-2019
  - A new value for the `as''` ice — `null`. Specifying `as"null"` is like specifying `pick"/dev/null" nocompletions`, i.e.: it disables the sourcing of the default script file of a plugin or snippet and also disables the installation of completions.
- 30-10-2019
  - A new ice `trigger-load''` — create a function that loads given plugin/snippet, with an option (to use it, precede the ice content with `!`) to automatically forward the call afterwards. Example use:

```
# Invoking the command `crasis' will load the plugin that
# provides the function `crasis', and it will be then
# immediately invoked with the same arguments
zinit ice trigger-load'!crasis'
zinit load z-shell/zinit-crasis
```

- 22-10-2019
  - A new ice `countdown` — causes an interruptable (by Ctrl-C) countdown 5…4…3…2…1…0 to be displayed before running the `atclone''`, `atpull''` and `make` ices.
- 21-10-2019
  - The `times` command has a new option `-m` — it shows the **moments** of the plugin load times — i.e.: how late after loading Zinit a plugin has been loaded.
- 20-10-2019
  - The `zinit` completion now completes also snippets! The command `snippet`, but also `delete`, `recall`, `edit`, `cd`, etc. all receive such completing.
  - The `ice` subcommand can now be skipped — just pass in the ices, e.g.:

    ```
    zinit atload"zicompinit; zicdreplay" blockf
    zinit light zsh-users/zsh-completions
    ```

  - The `compile` command is able to compile snippets.
  - The plugins that add their subdirectories into `$fpath` can be now `blockf`-ed — the functions located in the dirs will be correctly auto-loaded.
- 12-10-2019
  - Special value for the `id-as''` ice — `auto`. It sets the plugin/snippet ID automatically to the last component of its spec, e.g.:

    ```
    zinit ice id-as"auto"
    zinit load robobenklein/zinc
    ```

    will load the plugin as `id-as'zinc'`.
- 14-09-2019
  - There's a Vim plugin which extends syntax highlighting of zsh scripts with coloring of the Zinit commands. Project homepage.
- 13-09-2019
  - New ice `aliases` which loads plugin with the aliases mechanism enabled. Use for plugins that define **and use** aliases in their scripts.
- 11-09-2019

- New ice-mods `sh`,`bash`,`ksh`,`csh` that load plugins (and snippets) with the **sticky emulation** feature of Zsh — all functions defined within the plugin will automatically switch to the desired emulation mode before executing and switch back thereafter. In other words it is now possible to load e.g. bash plugins with Zinit, provided that the emulation level done by Zsh is sufficient, e.g.:

  ```
  zinit ice bash pick"bash_it.sh" \
          atinit"BASH_IT=${ZINIT[PLUGINS_DIR]}/Bash-it---bash-it" \
          atclone"yes n | ./install.sh"
  zinit load Bash-it/bash-it
  ```

  This script loads correctly thanks to the emulation, however it isn't functional because it uses `type -t …` to check if a function exists.

- 10-09-2019
  - A new ice-mod `reset''` that ivokes `git reset --hard` (or the provided command) before `git pull` and `atpull''` ice. It can be used it to implement altering (i.e. patching) of the plugin's files inside the `atpull''` ice — `git` will report no conflicts when doing `pull`, and the changes can be then again introduced by the `atpull''` ice.
  - Three new Zinit annexes (i.e. extensions):
    - z-a-man

      Generates man pages and code-documentation man pages from plugin's README.md and source files (the code documentation is obtained from Zshelldoc).

    - z-a-test

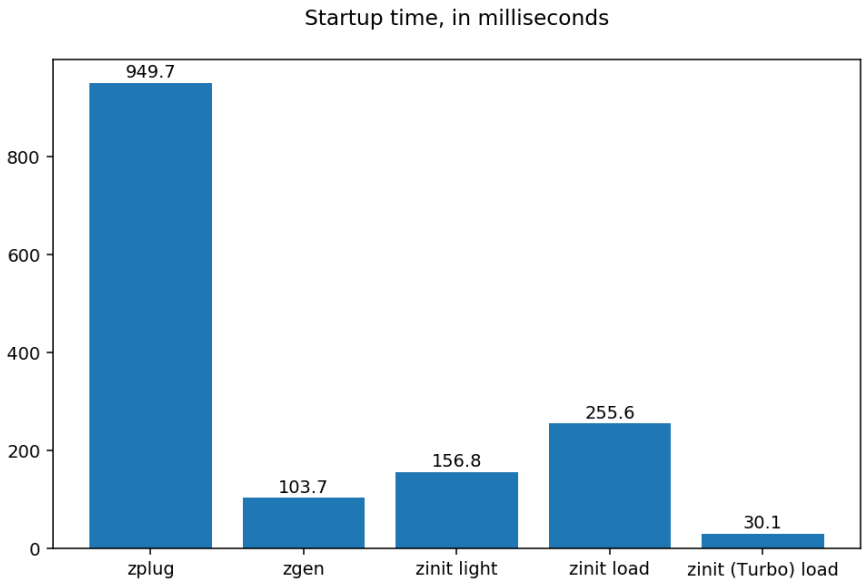
      Runs tests (if detected `test` target in a `Makefile` or any `*.zunit` files) on plugin installation and non-empty update.

    - z-a-patch-dl

      Allows easy download and applying of patches, to e.g. aid building a binary program equipped in the plugin.

  - A new variable is being recognized by the installation script: `$ZPLG_BIN_DIR_NAME`. It configures the directory within `$ZPLG_HOME` to which Zinit should be cloned.

To see the full history check the changelog.



Zinit is a flexible and fast Zshell plugin manager that will allow you to install everything from GitHub and other sites. Its characteristics are:

1. Zinit is currently the only plugin manager out there that provides Turbo mode which yields **50-80% faster Zsh startup** (i.e.: the shell will start up to **5** times faster!). Check out a speed comparison with other popular plugin managers here.

2. The plugin manager gives **reports** from plugin loadings describing what **aliases**, functions, **bindkeys**, Zle widgets, zstyles, **completions**, variables, `PATH` and `FPATH` elements a plugin has set up. This allows to quickly familiarize oneself with a new plugin and provides rich and easy to digest information which might be helpful on various occasions.

3. Supported is unloading of plugin and ability to list, (un)install and **selectively disable**, **enable** plugin's completions.

4. The plugin manager supports loading Oh My Zsh and Prezto plugins and libraries, however the implementation isn't framework specific and doesn't bloat the plugin manager with such code (more on this topic can be found on the Wiki, in the Introduction).

5. The system does not use `$FPATH`, loading multiple plugins doesn't clutter `$FPATH` with the same number of entries (e.g. `10`, `15` or more). Code is immune to `KSH_ARRAYS` and other options typically causing compatibility problems.

6. Zinit supports special, dedicated **packages** that offload the user from providing long and complex commands. See the Z-shell organization for a growing, complete list of Zinit packages and the Wiki page for an article about the feature.

7. Also, specialized Zinit extensions — called **annexes** — allow to extend the plugin manager with new commands, URL-preprocessors (used by e.g.: z-a-as-monitor annex), post-install and post-update hooks and much more. See the z-shell organization for a growing, complete list of available Zinit extensions and refer to the Wiki article for an introduction on creating your own annex.

Zinit Wiki

The information in this README is complemented by the Zinit Wiki. The README is an introductory overview of Zinit while the Wiki gives a complete information with examples. Make sure to read it to get the most out of Zinit.

Quick Start

Install

Automatic Installation (Recommended)

The easiest way to install Zinit is to execute:

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/z-
shell/zinit/main/doc/install.sh)"
```

This will install Zinit in `~/.zinit/bin`. `.zshrc` will be updated with three lines of code that will be added to the bottom. The lines will be sourcing `zinit.zsh` and setting up completion for command `zinit`.

After installing and reloading the shell compile Zinit with `zinit self-update`.

Manual Installation

To manually install Zinit clone the repo to e.g. `~/.zinit/bin`:

```
mkdir ~/.zinit
git clone https://github.com/z-shell/zinit.git ~/.zinit/bin
```

and source it from `.zshrc` (above compinit):

```
source ~/.zinit/bin/zinit.zsh
```

If you place the `source` below `compinit`, then add those two lines after the `source`:

```
autoload -Uz _zinit
(( ${+_comps} )) && _comps[zinit]=_zinit
```

Various paths can be customized, see section Customizing Paths.

After installing and reloading the shell compile Zinit with `zinit self-update`.

Usage

Introduction

Click here to read the introduction to Zinit. It explains basic usage and some of the more unique features of Zinit such as the Turbo mode. If you're new to Zinit we highly recommend you read it at least once.

Plugins and snippets

Plugins can be loaded using `load` or `light`.

```
zinit load  <repo/plugin> # Load with reporting/investigating.
zinit light <repo/plugin> # Load without reporting/investigating.
```

If you want to source local or remote files (using direct URL), you can do so with `snippet`.

```
zinit snippet <URL>
```

Such lines should be added to `.zshrc`. Snippets are cached locally, use `-f` option to download a fresh version of a snippet, or `zinit update {URL}`. Can also use `zinit update --all` to update all snippets (and plugins).

Example

```
# Plugin history-search-multi-word loaded with investigating.
zinit load z-shell/history-search-multi-word

# Two regular plugins loaded without investigating.
zinit light zsh-users/zsh-autosuggestions
zinit light z-shell/fast-syntax-highlighting

# Snippet
zinit snippet https://gist.githubusercontent.com/hightemp/5071909/raw/
```

Prompt(Theme) Example

This is powerlevel10k, pure, starship sample:

```
# Load powerlevel10k theme
zinit ice depth"1" # git clone depth
zinit light romkatv/powerlevel10k

# Load pure theme
zinit ice pick"async.zsh" src"pure.zsh" # with zsh-async library that's bundled
with it.
zinit light sindresorhus/pure

# Load starship theme
zinit ice as"command" from"gh-r" \ # `starship` binary as command, from github
release
```

```
          atclone"./starship init zsh > init.zsh; ./starship completions zsh >
_starship" \ # starship setup at clone(create init.zsh, completion)
          atpull"%atclone" src"init.zsh" # pull behavior same as clone, source
init.zsh
zinit light starship/starship
```

Upgrade Zinit and plugins

Zinit can be updated to `self-update` and plugins to `update`.

```
# Self update
zinit self-update

# Plugin update
zinit update

# Plugin parallel update
zinit update --parallel

# Increase the number of jobs in a concurrent-set to 40
zinit update --parallel 40
```

Turbo and lucid

Turbo and lucid are the most used options.

▶ **Turbo Mode**

Usually used as `zinit ice wait"<SECONDS>"`, let's use the previous example:

```
zinit ice wait    # wait is same wait"0"
zinit load z-shell/history-search-multi-word

zinit ice wait"2" # load after 2 seconds
zinit load z-shell/history-search-multi-word

zinit ice wait    # also be used in `light` and `snippet`
zinit snippet https://gist.githubusercontent.com/hightemp/5071909/raw/
```

▶ **Lucid**

Turbo mode is verbose, so you need an option for quiet.

You can use with `lucid`:

```
zinit ice wait lucid
zinit load z-shell/history-search-multi-word
```

F&A: What is `ice`?

`ice` is zinit's option command. The option melts like ice and is used only once. (more: Ice Modifiers)

Migration

▶ **Migration from Oh-My-ZSH**

Basic

```
zinit snippet <URL>         # Raw Syntax with URL
zinit snippet OMZ::<PATH>   # Shorthand OMZ/
(http://github.com/ohmyzsh/ohmyzsh/raw/master/)
zinit snippet OMZL::<PATH> # Shorthand OMZ/lib/
zinit snippet OMZT::<PATH> # Shorthand OMZ/themes/
zinit snippet OMZP::<PATH> # Shorthand OMZ/plugins/
```

Library

Importing the clipboard and termsupport Oh-My-Zsh Library Sample:

```
# Raw Syntax
zinit snippet https://github.com/ohmyzsh/ohmyzsh/blob/master/lib/clipboard.zsh
zinit snippet
https://github.com/ohmyzsh/ohmyzsh/blob/master/lib/termsupport.zsh

# OMZ Shorthand Syntax
zinit snippet OMZ::lib/clipboard.zsh
zinit snippet OMZ::lib/termsupport.zsh

# OMZL Shorthand Syntax
zinit snippet OMZL::clipboard.zsh
zinit snippet OMZL::termsupport.zsh
```

Theme

To use **themes** created for Oh My Zsh you might want to first source the `git` library there.

Then you can use the themes as snippets (`zinit snippet <file path or GitHub URL>`). Some themes require not only Oh My Zsh's Git **library**, but also Git **plugin** (error about `current_branch` may appear). Load this Git-plugin as single-file snippet directly from OMZ.

Most themes require `promptsubst` option (`setopt promptsubst` in `zshrc`), if it isn't set, then prompt will appear as something like: `... $(build_prompt) ...`.

You might want to suppress completions provided by the git plugin by issuing `zinit cdclear -q` (`-q` is for quiet) – see below **Ignoring Compdefs**.

To summarize:

```
## Oh My Zsh Setting
ZSH_THEME="robbyrussell"

## Zinit Setting
# Must Load OMZ Git library
zinit snippet OMZL::git.zsh

# Load Git plugin from OMZ
zinit snippet OMZP::git
zinit cdclear -q # <- forget completions provided up to this moment

setopt promptsubst

# Load Prompt
zinit snippet OMZT::robbyrussell
```

External Theme Sample: NicoSantangelo/Alpharized

```
## Oh My Zsh Setting
ZSH_THEME="alpharized"

## Zinit Setting
# Must Load OMZ Git library

zinit snippet OMZL::git.zsh

# Load Git plugin from OMZ
zinit snippet OMZP::git
zinit cdclear -q # <- forget completions provided up to this moment

setopt promptsubst

# Load Prompt
zinit light NicoSantangelo/Alpharized
```

F&A: Error occurs when loading OMZ's theme.

If the `git` library will not be loaded, then similar to following errors will be appearing:

```
........:1: command not found: git_prompt_status
........:1: command not found: git_prompt_short_sha
```

Plugin

If it consists of a single file, you can just load it.

```
## Oh-My-Zsh Setting
plugins=(
  git
  dotenv
  rake
  rbenv
  ruby
)

## Zinit Setting
zinit snippet OMZP::git
zinit snippet OMZP::dotenv
zinit snippet OMZP::rake
zinit snippet OMZP::rbenv
zinit snippet OMZP::ruby
```

Use `zinit ice svn` if multiple files require an entire subdirectory. Like gitfast, osx:

```
zinit ice svn
zinit snippet OMZP::gitfast

zinit ice svn
zinit snippet OMZP::osx
```

Use `zinit ice as"completion"` to directly add single file completion snippets. Like docker, fd:

```
zinit ice as"completion"
zinit snippet OMZP::docker/_docker

zinit ice as"completion"
zinit snippet OMZP::fd/_fd
```

You can see an extended explanation of Oh-My-Zsh setup in the Wiki

▶ **Migration from Prezto**

Basic

```
zinit snippet <URL>        # Raw Syntax with URL
zinit snippet PZT::<PATH>  # Shorthand PZT/ (https://github.com/sorin-
ionescu/prezto/tree/master/)
zinit snippet PZTM::<PATH> # Shorthand PZT/modules/
```

Modules

Importing the environment and terminal Prezto Modules Sample:

```
## Prezto Setting
zstyle ':prezto:load' pmodule 'environment' 'terminal'

## Zinit Setting
# Raw Syntax
zinit snippet https://github.com/sorin-
ionescu/prezto/blob/master/modules/environment/init.zsh
zinit snippet https://github.com/sorin-
ionescu/prezto/blob/master/modules/terminal/init.zsh

# PZT Shorthand Syntax
zinit snippet PZT::modules/environment
zinit snippet PZT::modules/terminal

# PZTM Shorthand Syntax
zinit snippet PZTM::environment
zinit snippet PZTM::terminal
```

Use `zinit ice svn` if multiple files require an entire subdirectory. Like docker, git:

```
zinit ice svn
zinit snippet PZTM::docker

zinit ice svn
zinit snippet PZTM::git
```

Use `zinit ice as"null"` if don't exist `*.plugin.zsh`, `init.zsh`, `*.zsh-theme*` files in module. Like archive:

```
zinit ice svn as"null"
zinit snippet PZTM::archive
```

Use `zinit ice atclone"git clone <repo> <location>"` if module have external module. Like completion:

```
zplugin ice svn blockf \ # use blockf to prevent any unnecessary additions to
fpath, as zinit manages fpath
            atclone"git clone --recursive https://github.com/zsh-users/zsh-
completions.git external"
zplugin snippet PZTM::completion
```

F&A: What is `zstyle`?

Read zstyle doc (more: What does `zstyle` do?).

▶ **Migration from Zgen**

Oh My Zsh

More reference: check **Migration from Oh-My-ZSH**

```
# Load ohmyzsh base
zgen oh-my-zsh
zinit snippet OMZL::<ALL OF THEM>

# Load ohmyzsh plugins
zgen oh-my-zsh <PATH>
zinit snippet OMZ::<PATH>
```

Prezto

More reference: check **Migration from Prezto**

```
# Load Prezto
zgen prezto
zinit snippet PZTM::<COMMENT's List> # environment terminal editor history
directory spectrum utility completion prompt

# Load prezto plugins
zgen prezto <modulename>
zinit snippet PZTM::<modulename>

# Load a repo as Prezto plugins
zgen pmodule <reponame> <branch>
zinit ice ver"<branch>"
zinit load <repo/plugin>

# Set prezto options
zgen prezto <modulename> <option> <value(s)>
zstyle ':prezto:<modulename>:' <option> <values(s)> # Set original prezto style
```

General

`location`: refer Selection of Files

```
zgen load <repo> [location] [branch]

zinit ice ver"[branch]"
zinit load <repo>
```

▶ **Migration from Zplug**

Basic

```
zplug <repo/plugin>, tag1:<option1>, tag2:<option2>

zinit ice tag1"<option1>" tag2"<option2>"
zinit load <repo/plugin>
```

Tag comparison

- as => as
- use => pick, src, multisrc
- ignore => None
- from => from
- at => ver
- rename-to => mv, cp
- dir => Selection(pick, ...) with rename
- if => if
- hook-build => atclone, atpull
- hook-load => atload
- frozen => None
- on => None
- defer => wait
- lazy => autoload
- depth => depth

More Examples

After installing Zinit you can start adding some actions (load some plugins) to ~/.zshrc, at bottom. Some examples:

```
# Load the pure theme, with zsh-async library that's bundled with it.
zinit ice pick"async.zsh" src"pure.zsh"
zinit light sindresorhus/pure

# A glance at the new for-syntax - load all of the above
# plugins with a single command. For more information see:
# https://z-shell.github.io/zinit/wiki/For-Syntax/
zinit for \
    light-mode  zsh-users/zsh-autosuggestions \
    light-mode  z-shell/fast-syntax-highlighting \
                z-shell/history-search-multi-word \
    light-mode pick"async.zsh" src"pure.zsh" \
                sindresorhus/pure

# Binary release in archive, from GitHub-releases page.
# After automatic unpacking it provides program "fzf".
zinit ice from"gh-r" as"program"
zinit light junegunn/fzf
```

```
# One other binary release, it needs renaming from `docker-compose-Linux-
x86_64`.
# This is done by ice-mod `mv'{from} -> {to}'. There are multiple packages per
# single version, for OS X, Linux and Windows - so ice-mod `bpick' is used to
# select Linux package - in this case this is actually not needed, Zinit will
# grep operating system name and architecture automatically when there's no
`bpick'.
zinit ice from"gh-r" as"program" mv"docker* -> docker-compose" bpick"*linux*"
zinit load docker/compose

# Vim repository on GitHub - a typical source code that needs compilation -
Zinit
# can manage it for you if you like, run `./configure` and other `make`, etc.
stuff.
# Ice-mod `pick` selects a binary program to add to $PATH. You could also
install the
# package under the path $ZPFX, see: http://z-
shell.github.io/zinit/wiki/Compiling-programs
zinit ice as"program" atclone"rm -f src/auto/config.cache; ./configure" \
    atpull"%atclone" make pick"src/vim"
zinit light vim/vim

# Scripts that are built at install (there's single default make target,
"install",
# and it constructs scripts by `cat'ing a few files). The make'' ice could also
be:
# `make"install PREFIX=$ZPFX"`, if "install" wouldn't be the only, default
target.
zinit ice as"program" pick"$ZPFX/bin/git-*" make"PREFIX=$ZPFX"
zinit light tj/git-extras

# Handle completions without loading any plugin, see "clist" command.
# This one is to be ran just once, in interactive session.
zinit creinstall %HOME/my_completions
```

```
# For GNU ls (the binaries can be gls, gdircolors, e.g. on OS X when installing
the
# coreutils package from Homebrew; you can also use
https://github.com/ogham/exa)
zinit ice atclone"dircolors -b LS_COLORS > c.zsh" atpull'%atclone' pick"c.zsh"
nocompile'!'
zinit light trapd00r/LS_COLORS
```

You can see an extended explanation of LS_COLORS in the Wiki.

```
# make'!...' -> run make before atclone & atpull
zinit ice as"program" make'!' atclone'./direnv hook zsh > zhook.zsh'
```

```
atpull'%atclone' src"zhook.zsh"
zinit light direnv/direnv
```

You can see an extended explanation of direnv in the Wiki.

If you're interested in more examples then check out the zinit-configs repository where users have uploaded their `~/.zshrc` and Zinit configurations. Feel free to submit your `~/.zshrc` there if it contains Zinit commands.

You can also check out the Gallery of Zinit Invocations for some additional examples.

Also, two articles on the Wiki present an example setup here and here.

How to Use

Ice Modifiers

Following `ice` modifiers are to be passed to `zinit ice ...` to obtain described effects. The word `ice` means something that's added (like ice to a drink) – and in Zinit it means adding modifier to a next `zinit` command, and also something that's temporary because it melts – and this means that the modification will last only for a **single** next `zinit` command.

Some Ice-modifiers are highlighted and clicking on them will take you to the appropriate Wiki page for an extended explanation.

You may safely assume a given ice works with both plugins and snippets unless explicitly stated otherwise.

Cloning Options

Modifier	Description
`proto`	Change protocol to `git`,`ftp`,`ftps`,`ssh`, `rsync`, etc. Default is `https`. **Does not work with snippets.**
`from`	Clone plugin from given site. Supported are `from"github"` (default), `..."github-rel"`, `..."gitlab"`, `..."bitbucket"`, `..."notabug"` (short names: `gh`, `gh-r`, `gl`, `bb`, `nb`). Can also be a full domain name (e.g. for GitHub enterprise). **Does not work with snippets.**
`ver`	Used with `from"gh-r"` (i.e. downloading a binary release, e.g. for use with `as"program"`) – selects which version to download. Default is latest, can also be explicitly `ver"latest"`. Works also with regular plugins, checkouts e.g. `ver"abranch"`, i.e. a specific version. **Does not work with snippets.**
`bpick`	Used to select which release from GitHub Releases to download, e.g. `zini ice from"gh-r" as"program" bpick"*Darwin*"; zini load docker/compose`. **Does not work with snippets.**
`depth`	Pass `--depth` to `git`, i.e. limit how much of history to download. **Does not work with snippets.**

Modifier	Description
`cloneopts`	Pass the contents of `cloneopts` to `git clone`. Defaults to `--recursive`. I.e.: change cloning options. Pass empty ice to disable recursive cloning. **Does not work with snippets.**
`pullopts`	Pass the contents of `pullopts` to `git pull` used when updating plugins. **Does not work with snippets.**
`svn`	Use Subversion for downloading snippet. GitHub supports `SVN` protocol, this allows to clone subdirectories as snippets, e.g. `zinit ice svn; zinit snippet OMZP::git`. Other ice `pick` can be used to select file to source (default are: `*.plugin.zsh`, `init.zsh`, `*.zsh-theme`). **Does not work with plugins.**

Selection of Files (To Source, …)

Modifier	Description
`pick`	Select the file to source, or the file to set as command (when using `snippet --command` or the ice `as"program"`); it is a pattern, alphabetically first matched file is being chosen; e.g. `zinit ice pick"*.plugin.zsh"; zinit load …`.
`src`	Specify additional file to source after sourcing main file or after setting up command (via `as"program"`). It is not a pattern but a plain file name.
`multisrc`	Allows to specify multiple files for sourcing, enumerated with spaces as the separators (e.g. `multisrc'misc.zsh grep.zsh'`) and also using brace-expansion syntax (e.g. `multisrc'{misc,grep}.zsh'`). Supports patterns.

Conditional Loading

Modifier	Description
`wait`	Postpone loading a plugin or snippet. For `wait'1'`, loading is done `1` second after prompt. For `wait'[[ ... ]]'`, `wait'(( ... ))'`, loading is done when given condition is meet. For `wait'!...'`, prompt is reset after load. Zsh can start 80% (i.e.: 5x) faster thanks to postponed loading. **Fact:** when `wait` is used without value, it works as `wait'0'`.
`load`	A condition to check which should cause plugin to load. It will load once, the condition can be still true, but will not trigger second load (unless plugin is unloaded earlier, see `unload` below). E.g.: `load'[[ $PWD = */github* ]]'`.
`unload`	A condition to check causing plugin to unload. It will unload once, then only if loaded again. E.g.: `unload'[[ $PWD != */github* ]]'`.
`cloneonly`	Don't load the plugin / snippet, only download it
`if`	Load plugin or snippet only when given condition is fulfilled, for example: `zinit ice if'[[ -n "$commands[otool]" ]]'; zinit load ...`.

Modifier	Description
has	Load plugin or snippet only when given command is available (in $PATH), e.g. zinit ice has'git' ...
subscribe / on-update-of	Postpone loading of a plugin or snippet until the given file(s) get updated, e.g. subscribe'{~/files-*,/tmp/files-*}'
trigger-load	Creates a function that loads the associated plugin/snippet, with an option (to use it, precede the ice content with !) to automatically forward the call afterwards, to a command of the same name as the function. Can obtain multiple functions to create – sparate with ;.

Plugin Output

Modifier	Description
silent	Mute plugin's or snippet's stderr & stdout. Also skip Loaded ... message under prompt for wait, etc. loaded plugins, and completion-installation messages.
lucid	Skip Loaded ... message under prompt for wait, etc. loaded plugins (a subset of silent).
notify	Output given message under-prompt after successfully loading a plugin/snippet. In case of problems with the loading, output a warning message and the return code. If starts with ! it will then always output the given message. Hint: if the message is empty, then it will just notify about problems.

Completions

Modifier	Description
blockf	Disallow plugin to modify fpath. Useful when a plugin wants to provide completions in traditional way. Zinit can manage completions and plugin can be blocked from exposing them.
nocompletions	Don't detect, install and manage completions for this plugin. Completions can be installed later with zinit creinstall {plugin-spec}.

Command Execution After Cloning, Updating or Loading

Modifier	Description
mv	Move file after cloning or after update (then, only if new commits were downloaded). Example: mv "fzf-* -> fzf". It uses -> as separator for old and new file names. Works also with snippets.
cp	Copy file after cloning or after update (then, only if new commits were downloaded). Example: cp "docker-c* -> dcompose". Ran after mv.

Modifier	Description
atclone	Run command after cloning, within plugin's directory, e.g. `zinit ice atclone"echo Cloned"`. Ran also after downloading snippet.
atpull	Run command after updating (**only if new commits are waiting for download**), within plugin's directory. If starts with "!" then command will be ran before `mv` & `cp` ices and before `git pull` or `svn update`. Otherwise it is ran after them. Can be `atpull'%atclone'`, to repeat `atclone` Ice-mod.
atinit	Run command after directory setup (cloning, checking it, etc.) of plugin/snippet but before loading.
atload	Run command after loading, within plugin's directory. Can be also used with snippets. Passed code can be preceded with `!`, it will then be investigated (if using `load`, not `light`).
`run-atpull`	Always run the atpull hook (when updating), not only when there are new commits to be downloaded.
`nocd`	Don't switch the current directory into the plugin's directory when evaluating the above ice-mods `atinit''`,`atload''`, etc.
make	Run `make` command after cloning/updating and executing `mv`, `cp`, `atpull`, `atclone` Ice mods. Can obtain argument, e.g. `make"install PREFIX=/opt"`. If the value starts with `!` then `make` is ran before `atclone`/`atpull`, e.g. `make'!'`.
`countdown`	Causes an interruptable (by Ctrl-C) countdown 5…4…3…2…1…0 to be displayed before executing `atclone''`,`atpull''` and `make` ices
`reset`	Invokes `git reset --hard HEAD` for plugins or `svn revert` for SVN snippets before pulling any new changes. This way `git` or `svn` will not report conflicts if some changes were done in e.g.: `atclone''` ice. For file snippets and `gh-r` plugins it invokes `rm -rf *`.

Sticky-Emulation Of Other Shells

Modifier	Description
`sh`, `!sh`	Source the plugin's (or snippet's) script with `sh` emulation so that also all functions declared within the file will get a *sticky* emulation assigned – when invoked they'll execute also with the `sh` emulation set-up. The `!sh` version switches additional options that are rather not important from the portability perspective.
`bash`, `!bash`	The same as `sh`, but with the `SH_GLOB` option disabled, so that Bash regular expressions work.
`ksh`, `!ksh`	The same as `sh`, but emulating `ksh` shell.
`csh`, `!csh`	The same as `sh`, but emulating `csh` shell.

Others

Modifier	Description
`as`	Can be `as"program"` (also the alias: `as"command"`), and will cause to add script/program to `$PATH` instead of sourcing (see `pick`). Can also be `as"completion"` – use with plugins or snippets in whose only underscore-starting `_*` files you are interested in. The third possible value is `as"null"` – a shorthand for `pick"/dev/null" nocompletions` – i.e.: it disables the default script-file sourcing and also the installation of completions.
`id-as`	Nickname a plugin or snippet, to e.g. create a short handler for long-url snippet.
`compile`	Pattern (+ possible `{...}` expansion, like `{a/*,b*}`) to select additional files to compile, e.g. `compile"(pure|async).zsh"` for `sindresorhus/pure`.
`nocompile`	Don't try to compile `pick`-pointed files. If passed the exclamation mark (i.e. `nocompile'!'`), then do compile, but after `make''` and `atclone''` (useful if Makefile installs some scripts, to point `pick''` at the location of their installation).
`service`	Make following plugin or snippet a *service*, which will be ran in background, and only in single Zshell instance. See #zinit-service topic.
`reset-prompt`	Reset the prompt after loading the plugin/snippet (by issuing `zle .reset-prompt`). Note: normally it's sufficient to precede the value of `wait''` ice with `!`.
`bindmap`	To hold `;`-separated strings like `Key(s)A -> Key(s)B`, e.g. `^R -> ^T; ^A -> ^B`. In general, `bindmap''`changes bindings (done with the `bindkey` builtin) the plugin does. The example would cause the plugin to map Ctrl-T instead of Ctrl-R, and Ctrl-B instead of Ctrl-A. **Does not work with snippets.**
`trackbinds`	Shadow but only `bindkey` calls even with `zinit light ...`, i.e. even with investigating disabled (fast loading), to allow `bindmap` to remap the key-binds. The same effect has `zinit light -b ...`, i.e. additional `-b` option to the `light`-subcommand. **Does not work with snippets.**
`wrap-track`	Takes a `;`-separated list of function names that are to be investigated (meaning gathering report and unload data) **once** during execution. It works by wrapping the functions with a investigating-enabling and disabling snippet of code. In summary, `wrap-track` allows to extend the investigating beyond the moment of loading of a plugin. Example use is to `wrap-track` a precmd function of a prompt (like `_p9k_precmd()` of powerlevel10k) or other plugin that *postpones its initialization till the first prompt* (like e.g.: zsh-autosuggestions). **Does not work with snippets.**
`aliases`	Load the plugin with the aliases mechanism enabled. Use with plugins that define **and use** aliases in their scripts.
`light-mode`	Load the plugin without the investigating, i.e.: as if it would be loaded with the `light` command. Useful for the for-syntax, where there is no `load` nor `light` subcommand

Modifier	Description
`extract`	Performs archive extraction supporting multiple formats like `zip`, `tar.gz`, etc. and also notably OS X `dmg` images. If it has no value, then it works in the *auto* mode – it automatically extracts all files of known archive extensions IF they aren't located deeper than in a sub-directory (this is to prevent extraction of some helper archive files, typically located somewhere deeper in the tree). If no such files will be found, then it extracts all found files of known **type** – the type is being read by the `file` Unix command. If not empty, then takes names of the files to extract. Refer to the Wiki page for further information.
`subst`	Substitute the given string into another string when sourcing the plugin script, e.g.: `zinit subst'autoload → autoload -Uz' …`.
`autoload`	Autoload the given functions (from their files). Equvalent to calling `atinit'autoload the-function'`. Supports renaming of the function – pass `'… → new-name'` or `'… -> new-name'`, e.g.: `zinit autoload'fun → my-fun; fun2 → my-fun2'`.

Order of Execution

Order of execution of related Ice-mods: `atinit` -> `atpull!` -> `make'!!'` -> `mv` -> `cp` -> `make!` -> `atclone`/`atpull` -> `make` -> `(plugin script loading)` -> `src` -> `multisrc` -> `atload`.

Zinit Commands

Following commands are passed to `zinit ...` to obtain described effects.

Help

Command	Description
`-h, --help, help`	Usage information.
`man`	Manual.

Loading and Unloading

Command	Description
`load {plg-spec}`	Load plugin, can also receive absolute local path.
`light [-b] {plg-spec}`	Light plugin load, without reporting/investigating. `-b` – investigate `bindkey`-calls only. There's also `light-mode` ice which can be used to induce the no-investigating (i.e.: *light*) loading, regardless of the command used.
`unload [-q] {plg-spec}`	Unload plugin loaded with `zinit load ...`. `-q` – quiet.

Command	Description
`snippet [-f] {url}`	Source local or remote file (by direct URL). `-f` – don't use cache (force redownload). The URL can use the following shorthands: `PZT::` (Prezto), `PZTM::` (Prezto module), `OMZ::` (Oh My Zsh), `OMZP::` (OMZ plugin), `OMZL::` (OMZ library), `OMZT::` (OMZ theme), e.g.: `PZTM::environment`, `OMZP::git`, etc.

Completions Management

Command	Description
`clist [columns]`, `completions [columns]`	List completions in use, with *`columns`* completions per line. `zpl clist 5` will for example print 5 completions per line. Default is 3.
`cdisable {cname}`	Disable completion `cname`.
`cenable {cname}`	Enable completion `cname`.
`creinstall [-q] [-Q] {plg-spec}`	Install completions for plugin, can also receive absolute local path. `-q` – quiet. `-Q` - quiet all.
`cuninstall {plg-spec}`	Uninstall completions for plugin.
`csearch`	Search for available completions from any plugin.
`compinit`	Refresh installed completions.
`cclear`	Clear stray and improper completions.
`cdlist`	Show compdef replay list.
`cdreplay [-q]`	Replay compdefs (to be done after compinit). `-q` – quiet.
`cdclear [-q]`	Clear compdef replay list. `-q` – quiet.

Tracking of the Active Session

Command	Description
`dtrace, dstart`	Start investigating what's going on in session.
`dstop`	Stop investigating what's going on in session.
`dunload`	Revert changes recorded between dstart and dstop.
`dreport`	Report what was going on in session.
`dclear`	Clear report of what was going on in session.

Reports and Statistics

Command	Description
`times [-s] [-m]`	Statistics on plugin load times, sorted in order of loading. `-s` – use seconds instead of milliseconds. `-m` – show plugin loading moments.

Command	Description
`zstatus`	Overall Zinit status.
`report {plg-spec}|--all`	Show plugin report. `--all` – do it for all plugins.
`loaded [keyword], list [keyword]`	Show what plugins are loaded (filter with 'keyword').
`ls`	List snippets in formatted and colorized manner. Requires **tree** program.
`status {plg-spec}|URL|--all`	Git status for plugin or svn status for snippet. `--all` – do it for all plugins and snippets.
`recently [time-spec]`	Show plugins that changed recently, argument is e.g. 1 month 2 days.
`bindkeys`	Lists bindkeys set up by each plugin.

Compiling

Command	Description
`compile {plg-spec}|--all`	Compile plugin. `--all` – compile all plugins.
`uncompile {plg-spec}|--all`	Remove compiled version of plugin. `--all` – do it for all plugins.
`compiled`	List plugins that are compiled.

Other

Command	Description
`self-update`	Updates and compiles Zinit.
`update [-q] [-r] {plg-spec}|URL|--all`	Git update plugin or snippet. `--all` – update all plugins and snippets. `-q` – quiet. `-r` | `--reset` – run `git reset --hard` / `svn revert` before pulling changes.
`ice <ice specification>`	Add ice to next command, argument is e.g. from"gitlab".
`delete {plg-spec}|URL|--clean|--all`	Remove plugin or snippet from disk (good to forget wrongly passed ice-mods). `--all` – purge. `--clean` – delete plugins and snippets that are not loaded.
`cd {plg-spec}`	Cd into plugin's directory. Also support snippets if fed with URL.
`edit {plg-spec}`	Edit plugin's file with $EDITOR.

Command	Description
`glance {plg-spec}`	Look at plugin's source (pygmentize, {,source-}highlight).
`stress {plg-spec}`	Test plugin for compatibility with set of options.
`changes {plg-spec}`	View plugin's git log.
`create {plg-spec}`	Create plugin (also together with GitHub repository).
`srv {service-id} [cmd]`	Control a service, command can be: stop,start,restart,next,quit; `next` moves the service to another Zshell.
`recall {plg-spec}|URL`	Fetch saved ice modifiers and construct `zinit ice ...` command.
`env-whitelist [-v] [-h] {env..}`	Allows to specify names (also patterns) of variables left unchanged during an unload. `-v` – verbose.
`module`	Manage binary Zsh module shipped with Zinit, see `zinit module help`.
`add-fpath|fpath [-f|--front] {plg-spec} [subdirectory]`	Adds given plugin (not yet snippet) directory to `$fpath`. If the second argument is given, it is appended to the directory path. If the option `-f`/`--front` is given, the directory path is prepended instead of appended to `$fpath`. The `{plg-spec}` can be absolute path, i.e.: it's possible to also add regular directories.
`run [-l] [plugin] {command}`	Runs the given command in the given plugin's directory. If the option `-l` will be given then the plugin should be skipped – the option will cause the previous plugin to be reused.

Updating Zinit and Plugins

To update Zinit issue `zinit self-update` in the command line.

To update all plugins and snippets, issue `zinit update`. If you wish to update only a single plugin/snippet instead issue `zinit update NAME_OF_PLUGIN`. A list of commits will be shown:

Some plugins require performing an action each time they're updated. One way you can do this is by using the `atpull` ice modifier. For example, writing `zinit ice atpull'./configure'` before loading a plugin will execute `./configure` after a successful update. Refer to Ice Modifiers for more information.

The ice modifiers for any plugin or snippet are stored in their directory in a `._zinit` subdirectory, hence the plugin doesn't have to be loaded to be correctly updated. There's one other file created there, `.zinit_lstupd` – it holds the log of the new commits pulled-in in the last update.

Calling `compinit` Without Turbo Mode

With no Turbo mode in use, compinit can be called normally, i.e.: as `autoload compinit; compinit`. This should be done after loading of all plugins and before possibly calling `zinit cdreplay`.

The `cdreplay` subcommand is provided to re-play all catched `compdef` calls. The `compdef` calls are used to define a completion for a command. For example, `compdef _git git` defines that the `git` command should be completed by a `_git` function.

The `compdef` function is provided by `compinit` call. As it should be called later, after loading all of the plugins, Zinit provides its own `compdef` function that catches (i.e.: records in an array) the arguments of the call, so that the loaded plugins can freely call `compdef`. Then, the `cdreplay` (*compdef-replay*) can be used, after `compinit` will be called (and the original `compdef` function will become available), to execute all detected `compdef` calls. To summarize:

```
source ~/.zinit/bin/zinit.zsh

zinit load "some/plugin"
...
compdef _gnu_generic fd  # this will be intercepted by Zinit, because as the
compinit
                         # isn't yet loaded, thus there's no such function
`compdef'; yet
                         # Zinit provides its own `compdef' function which
saves the
                         # completion-definition for later possible re-run with
`zinit
                         # cdreplay' or `zicdreplay' (the second one can be
used in hooks
                         # like atload'', atinit'', etc.)
...
zinit load "other/plugin"

autoload -Uz compinit
compinit

zinit cdreplay -q   # -q is for quiet; actually run all the `compdef's saved
before
                    #`compinit` call (`compinit' declares the `compdef'
function, so
                    # it cannot be used until `compinit' is ran; Zinit solves
this
                    # via intercepting the `compdef'-calls and storing them for
later
                    # use with `zinit cdreplay')
```

This allows to call compinit once. Performance gains are huge, example shell startup time with double `compinit`: **0.980** sec, with `cdreplay` and single `compinit`: **0.156** sec.

Calling `compinit` With Turbo Mode

If you load completions using `wait''` Turbo mode then you can add `atinit'zicompinit'` to syntax-highlighting plugin (which should be the last one loaded, as their (2 projects, z-sy-h & f-sy-h) documentation state), or `atload'zicompinit'` to last completion-related plugin. `zicompinit` is a function that just runs `autoload compinit; compinit`, created for convenience. There's also `zicdreplay` which will replay any caught compdefs so you can also do: `atinit'zicompinit; zicdreplay'`, etc. Basically, the whole topic is the same as normal `compinit` call, but it is done in `atinit` or `atload` hook of the last related plugin with use of the helper functions (`zicompinit`,`zicdreplay` & `zicdclear` – see below for explanation of the last one). To summarize:

```
source ~/.zinit/bin/zinit.zsh

# Load using the for-syntax
zinit wait lucid for \
    "some/plugin"
zinit wait lucid for \
    "other/plugin"

zinit wait lucid atload"zicompinit; zicdreplay" blockf for \
    zsh-users/zsh-completions
```

Ignoring Compdefs

If you want to ignore compdefs provided by some plugins or snippets, place their load commands before commands loading other plugins or snippets, and issue `zinit cdclear` (or `zicdclear`, designed to be used in hooks like `atload''`):

```
source ~/.zinit/bin/zinit.zsh
zinit snippet OMZP::git
zinit cdclear -q # <- forget completions provided by Git plugin

zinit load "some/plugin"
...
zinit load "other/plugin"

autoload -Uz compinit
compinit
zinit cdreplay -q # <- execute compdefs provided by rest of plugins
zinit cdlist # look at gathered compdefs
```

The `cdreplay` is important if you use plugins like `OMZP::kubectl` or `asdf-vm/asdf`, because these plugins call `compdef`.

Disabling System-Wide `compinit` Call (Ubuntu)

On Ubuntu users might get surprised that e.g. their completions work while they didn't call `compinit` in their `.zshrc`. That's because the function is being called in `/etc/zshrc`. To disable this call – what is

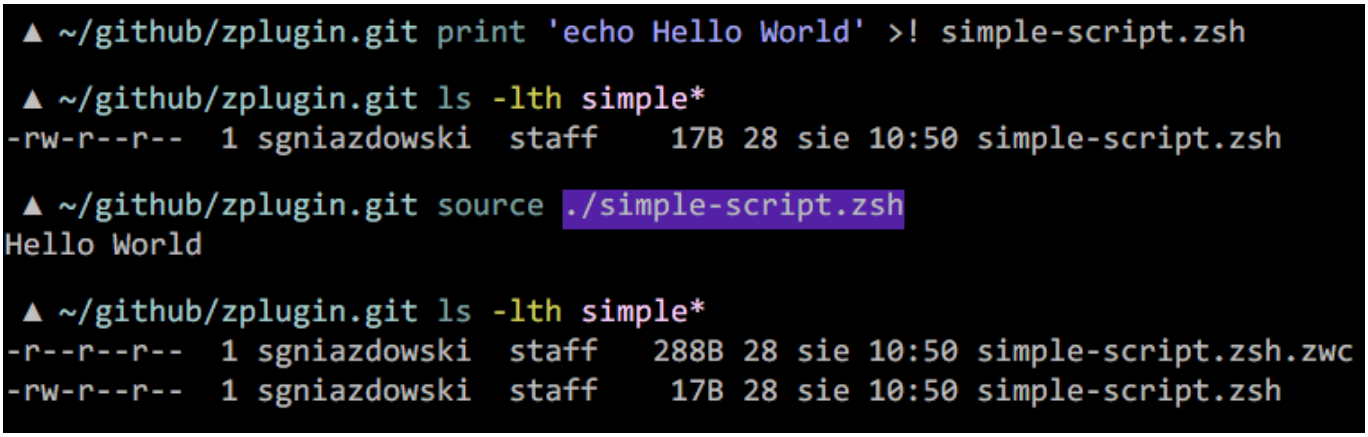
needed to avoid the slowdown and if user loads any completion-equipped plugins, i.e. almost on 100% – add the following lines to `~/.zshenv`:

```
# Skip the not really helping Ubuntu global compinit
skip_global_compinit=1
```

Zinit Module

Motivation

The module is a binary Zsh module (think about `zmodload` Zsh command, it's that topic) which transparently and automatically **compiles sourced scripts**. Many plugin managers do not offer compilation of plugins, the module is a solution to this. Even if a plugin manager does compile plugin's main script (like Zinit does), the script can source smaller helper scripts or dependency libraries (for example, the prompt `geometry-zsh/geometry` does that) and there are very few solutions to that, which are demanding (e.g. specifying all helper files in plugin load command and investigating updates to the plugin – in Zinit case: by using `compile` ice-mod).



Installation

Without Zinit

To install just the binary Zinit module **standalone** (Zinit is not needed, the module can be used with any other plugin manager), execute:

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/z-
shell/zinit/main/doc/mod-install.sh)"
```

This script will display what to add to `~/.zshrc` (2 lines) and show usage instructions.

With Zinit

Zinit users can build the module by issuing following command instead of running above `mod-install.sh` script (the script is for e.g. `zgen` users or users of any other plugin manager):

```
zinit module build
```

This command will compile the module and display instructions on what to add to `~/.zshrc`.

Measuring Time of `source`s

Besides the compilation-feature, the module also measures **duration** of each script sourcing. Issue `zpmod source-study` after loading the module at top of `~/.zshrc` to see a list of all sourced files with the time the sourcing took in milliseconds on the left. This feature allows to profile the shell startup. Also, no script can pass-through that check and you will obtain a complete list of all loaded scripts, like if Zshell itself was investigating this. The list can be surprising.

Debugging

To enable debug messages from the module set:

```
typeset -g ZPLG_MOD_DEBUG=1
```

Hints and Tips

Customizing Paths

Following variables can be set to custom values, before sourcing Zinit. The previous global variables like `$ZPLG_HOME` have been removed to not pollute the namespace – there's single `$ZINIT` hash instead of `8` string variables. Please update your dotfiles.

```
declare -A ZINIT  # initial Zinit's hash definition, if configuring before
loading Zinit, and then:
```

Hash Field	Description
ZINIT[BIN_DIR]	Where Zinit code resides, e.g.: "~/.zinit/bin"
ZINIT[HOME_DIR]	Where Zinit should create all working directories, e.g.: "~/.zinit"
ZINIT[PLUGINS_DIR]	Override single working directory – for plugins, e.g. "/opt/zsh/zinit/plugins"
ZINIT[COMPLETIONS_DIR]	As above, but for completion files, e.g. "/opt/zsh/zinit/root_completions"
ZINIT[SNIPPETS_DIR]	As above, but for snippets
ZINIT[ZCOMPDUMP_PATH]	Path to `.zcompdump` file, with the file included (i.e. its name can be different)

Hash Field	Description
ZINIT[COMPINIT_OPTS]	Options for `compinit` call (i.e. done by `zicompinit`), use to pass -C to speed up loading
ZINIT[MUTE_WARNINGS]	If set to `1`, then mutes some of the Zinit warnings, specifically the `plugin already registered` warning
ZINIT[OPTIMIZE_OUT_DISK_ACCESSES]	If set to `1`, then Zinit will skip checking if a Turbo-loaded object exists on the disk. By default Zinit skips Turbo for non-existing objects (plugins or snippets) to install them before the first prompt – without any delays, during the normal processing of `zshrc`. This option can give a performance gain of about 10 ms out of 150 ms (i.e.: Zsh will start up in 140 ms instead of 150 ms).

There is also `$ZPFX`, set by default to `~/.zinit/polaris` – a directory where software with `Makefile`, etc. can be pointed to, by e.g. `atclone'./configure --prefix=$ZPFX'`.

Non-GitHub (Local) Plugins

Use `create` subcommand with user name `_local` (the default) to create plugin's skeleton in `$ZINIT[PLUGINS_DIR]`. It will be not connected with GitHub repository (because of user name being `_local`). To enter the plugin's directory use `cd` command with just plugin's name (without `_local`, it's optional).

If user name will not be `_local`, then Zinit will create repository also on GitHub and setup correct repository origin.

Extending Git

There are several projects that provide git extensions. Installing them with Zinit has many benefits:

- all files are under `$HOME` – no administrator rights needed,
- declarative setup (like Chef or Puppet) – copying `.zshrc` to different account brings also git-related setup,
- easy update by e.g. `zinit update --all`.

Below is a configuration that adds multiple git extensions, loaded in Turbo mode, 1 second after prompt, with use of the Bin-Gem-Node annex:

```
zinit as"null" wait"1" lucid for \
    sbin    Fakerr/git-recall \
    sbin    cloneopts paulirish/git-open \
    sbin    paulirish/git-recent \
    sbin    davidosomething/git-my \
    sbin atload"export _MENU_THEME=legacy" \
            arzzen/git-quick-stats \
    sbin    iwata/git-now \
    make"PREFIX=$ZPFX install" \
```

```
        tj/git-extras \
    sbin"bin/git-dsf;bin/diff-so-fancy" \
        z-shell/zsh-diff-so-fancy \
    sbin"git-url;git-guclone" make"GITURL_NO_CGITURL=1" \
        z-shell/git-url
```

Target directory for installed files is `$ZPFX` (`~/.zinit/polaris` by default).