

📄 junegunn / **fzf** Public

🌸 A command-line fuzzy finder

📄 MIT license

☆ 45.7k stars    🍴 2k forks

★ Starred

👁 Watch ▼

Code

Issues

246

Pull requests

52

Discussions

Actions

Wiki

Security

Insights

🔑 master ▼

...

加速 ▼

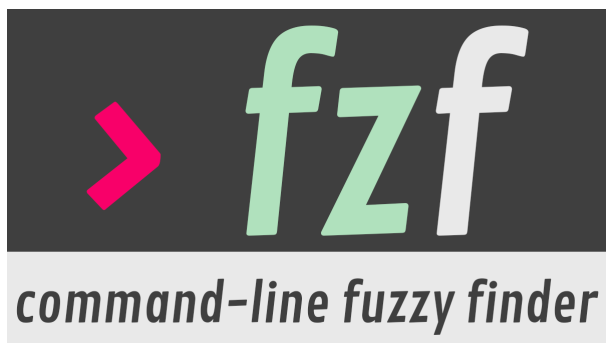


jzacsh ...

✓ yesterday



[View code](#)



🔄 Test fzf on Linux passing

fzf is a general-purpose command-line fuzzy finder.

```
src/util/eventbox_test.go
src/curses/curses_test.go
src/util/atomicbool.go
src/util/util_test.go
src/tokenizer_test.go
src/chunklist_test.go
src/algo/algo_test.go
src/util/eventbox.go
src/curses/curses.go
src/pattern_test.go
src/options_test.go
src/history_test.go
src/reader_test.go
src/merger_test.go
src/cache_test.go
src/util/util.go
src/tokenizer.go
src/item_test.go
src/constants.go
src/chunklist.go
src/ansi_test.go
src/algo/algo.go
src/terminal.go
src/fzf/main.go
src/pattern.go
> src/options.go
src/matcher.go
src/history.go
src/reader.go
src/merger.go
src/cache.go
src/item.go
src/core.go
src/ansi.go
35/63
> .go$
```

```
package fzf

import (
    "fmt"
    "os"
    "regexp"
    "strconv"
    "strings"
    "unicode/utf8"

    "github.com/junegunn/fzf/src/curses"
    "github.com/junegunn/go-shellwords"
)

const usage = `usage: fzf [options]

Search
-X, --extended      Extended-search mode
                     (enabled by default; +x or --no-extended to disable)
-e, --exact          Enable Exact-match
-i                  Case-insensitive match (default: smart-case match)
+i                  Case-sensitive match
-n, --nth=N[,..]    Comma-separated list of field index expressions
                     for limiting search scope. Each can be a non-zero
                     integer or a range expression ([BEGIN]..[END]).
--with-nth=N[,..]   Transform the presentation of each line using
                     field index expressions
-d, --delimiter=STR Field delimiter regex (default: AWK-style)
+s, --no-sort        Do not sort the result
--tac               Reverse the order of the input
--tiebreak=CRI[,..] Comma-separated list of sort criteria to apply
                     when the scores are tied [length|begin|end|index]
                     (default: length)
```

It's an interactive Unix filter for command-line that can be used with any list; files, command history, processes, hostnames, bookmarks, git commits, etc.

## Pros

---

- Portable, no dependencies
- Blazingly fast
- The most comprehensive feature set
- Flexible layout
- Batteries included
  - Vim/Neovim plugin, key bindings, and fuzzy auto-completion

## Table of Contents

---

- [Installation](#)
  - [Using Homebrew](#)
  - [Using git](#)
  - [Using Linux package managers](#)
  - [Windows](#)
  - [As Vim plugin](#)
- [Upgrading fzf](#)
- [Building fzf](#)
- [Usage](#)

- Using the finder
- Layout
- Search syntax
- Environment variables
- Options
- Demo
- Examples
- `fzf-tmux` script
- Key bindings for command-line
- Fuzzy completion for bash and zsh
  - Files and directories
  - Process IDs
  - Host names
  - Environment variables / Aliases
  - Settings
  - Supported commands
  - Custom fuzzy completion
- Vim plugin
- Advanced topics
  - Performance
  - Executing external programs
  - Reloading the candidate list
    - 1. Update the list of processes by pressing CTRL-R
    - 2. Switch between sources by pressing CTRL-D or CTRL-F
    - 3. Interactive ripgrep integration
  - Preview window
- Tips
  - Respecting `.gitignore`
  - Fish shell
- Related projects
- License

## Installation

---

fzf project consists of the following components:

- `fzf` executable
- `fzf-tmux` script for launching fzf in a tmux pane

- Shell extensions
  - Key bindings ( CTRL-T , CTRL-R , and ALT-C ) (bash, zsh, fish)
  - Fuzzy auto-completion (bash, zsh)
- Vim/Neovim plugin

You can [download fzf executable](#) alone if you don't need the extra stuff.

## Using Homebrew

You can use [Homebrew](#) (on macOS or Linux) to install fzf.

```
brew install fzf

# To install useful key bindings and fuzzy completion:
$(brew --prefix)/opt/fzf/install
```

fzf is also available [via MacPorts](#): `sudo port install fzf`

## Using git

Alternatively, you can "git clone" this repository to any directory and run [install](#) script.

```
git clone --depth 1 https://github.com/junegunn/fzf.git ~/.fzf
~/.fzf/install
```

## Using Linux package managers

Package Manager	Linux Distribution	Command
APK	Alpine Linux	<code>sudo apk add fzf</code>
APT	Debian 9+/Ubuntu 19.10+	<code>sudo apt-get install fzf</code>
Conda		<code>conda install -c conda-forge fzf</code>
DNF	Fedora	<code>sudo dnf install fzf</code>
Nix	NixOS, etc.	<code>nix-env -iA nixpkgs.fzf</code>
Pacman	Arch Linux	<code>sudo pacman -S fzf</code>
pkg	FreeBSD	<code>pkg install fzf</code>
pkgin	NetBSD	<code>pkgin install fzf</code>
pkg_add	OpenBSD	<code>pkg_add fzf</code>

## ☰ README.md

Xbps	Void Linux	<code>sudo xbps-install -S fzf</code>
Zypper	openSUSE	<code>sudo zypper install fzf</code>

⚠ Key bindings (CTRL-T / CTRL-R / ALT-C) and fuzzy auto-completion may not be enabled by default.

Refer to the package documentation for more information. (e.g. `apt-cache show fzf` )

Packaging status	
Alpine Linux 3.8	0.17.3
Alpine Linux 3.9	0.17.5
Alpine Linux 3.10	0.18.0
Alpine Linux 3.11	0.19.0
Alpine Linux 3.12	0.21.1
Alpine Linux 3.13	0.25.0
Alpine Linux 3.14	0.27.2
Alpine Linux 3.15	0.28.0
Alpine Linux 3.16	0.30.0
Alpine Linux Edge	0.30.0
ALT Linux p9	0.21.1
ALT Linux p10	0.27.2
ALT Sisyphus	0.30.0
AOSC	0.30.0
Arch	0.30.0
Arch Linux 32 i486	0.23.1
Arch Linux 32 i686	0.29.0
Arch Linux 32 pentium4	0.30.0
Arch Linux ARM aarch64	0.30.0
Arch Linux ARM armv7h	0.30.0
AUR	0.28.0.r2....
Artix	0.30.0
Carbs Linux	0.30.0
Chocolatey	0.30.0
CRUX 3.5	0.24.4
CRUX 3.6	0.30.0
Cygwin	0.12.1
Debian 9 Backports	0.17.5
Debian 10	0.17.5
Debian 11	0.24.3
Debian 12	0.30.0
Debian Unstable	0.30.0
Deepin	0.17.5
Devuan 3.0	0.17.5
Devuan 4.0	0.24.3
Devuan Unstable	0.30.0
distri	v0.0.0-202...
DPorts	0.27.0
Fedora 26	0.17.3
Fedora 27	0.17.5
Fedora 28	0.17.5
Fedora 29	0.18.0
Fedora 30	0.20.0
Fedora 31	0.24.3
Fedora 32	0.26.0
Fedora 33	0.28.0
Fedora 34	0.29.0
Fedora 35	0.29.0
Fedora 36	0.30.0
Fedora Rawhide	0.30.0
FreeBSD Ports	0.30.0

Funtoo 1.4	0.30.0
Gentoo	0.30.0
GNU Guix	0.25.0
Homebrew	0.30.0
Kali Linux Rolling	0.30.0
LiGurOS stable	0.30.0
LiGurOS develop	0.30.0
MacPorts	0.30.0
Manjaro Stable	0.30.0
Manjaro Testing	0.30.0
Manjaro Unstable	0.30.0
MidnightBSD mports	0.19.0
MPR	0.27.2.2.g...
MSYS2 mingw	0.30.0
nixpkgs stable 21.05	0.27.1
nixpkgs stable 21.11	0.28.0
nixpkgs stable 22.05	0.30.0
nixpkgs unstable	0.30.0
OpenBSD Ports	0.24.1
OpenPKG	0.30.0.202...
openSUSE Leap 15.2	0.21.1
openSUSE Leap 15.3	0.21.1
openSUSE Leap 15.4	0.28.0
openSUSE Tumbleweed	0.30.0
Parabola	0.30.0
Pardus 19	0.17.5
Pardus 21	0.24.3
Parrot	0.24.3
pkgsrc current	0.29.0
PLD Linux	0.30.0
PureOS Amber	0.17.5
PureOS landing	0.24.3
Raspbian Oldstable	0.17.5
Raspbian Stable	0.24.3
Raspbian Testing	0.30.0
Scoop	0.30.0
SlackBuilds	0.30.0
Solus	0.30.0
Spack	0.22.0
T2 SDE	0.30.0
Termux	0.30.0
Trisquel 10.0	0.20.0
Ubuntu 20.04	0.20.0
Ubuntu 21.04	0.24.3
Ubuntu 21.10	0.24.3
Ubuntu 22.04	0.29.0
Ubuntu 22.10	0.30.0
Void Linux x86_64	0.30.0
Wikidata	0.30.0

## Windows

Pre-built binaries for Windows can be downloaded [here](#). fzf is also available via [Chocolatey](#) and [Scoop](#):

Package manager	Command
Chocolatey	<code>choco install fzf</code>
Scoop	<code>scoop install fzf</code>

Known issues and limitations on Windows can be found on [the wiki page](#).

## As Vim plugin

If you use [vim-plug](#), add this line to your Vim configuration file:

```
Plug 'junegunn/fzf', { 'do': { -> fzf#install() } }
```

`fzf#install()` makes sure that you have the latest binary, but it's optional, so you can omit it if you use a plugin manager that doesn't support hooks.

For more installation options, see [README-VIM.md](#).

## Upgrading fzf

---

fzf is being actively developed, and you might want to upgrade it once in a while. Please follow the instruction below depending on the installation method used.

- git: `cd ~/.fzf && git pull && ./install`
- brew: `brew update; brew upgrade fzf`
- macports: `sudo port upgrade fzf`
- chocolatey: `choco upgrade fzf`
- vim-plug: `:PlugUpdate fzf`

## Building fzf

---

See [BUILD.md](#).

## Usage

---

fzf will launch interactive finder, read the list from STDIN, and write the selected item to STDOUT.

```
find * -type f | fzf > selected
```



Without STDIN pipe, fzf will use find command to fetch the list of files excluding hidden ones. (You can override the default command with `FZF_DEFAULT_COMMAND` )

```
vim $(fzf)
```

## Using the finder

- `CTRL-K` / `CTRL-J` (or `CTRL-P` / `CTRL-N` ) to move cursor up and down
- `Enter` key to select the item, `CTRL-C` / `CTRL-G` / `ESC` to exit
- On multi-select mode ( `-m` ), `TAB` and `Shift-TAB` to mark multiple items
- Emacs style key bindings
- Mouse: scroll, click, double-click; shift-click and shift-scroll on multi-select mode

## Layout

fzf by default starts in fullscreen mode, but you can make it start below the cursor with `--height` option.

```
vim $(fzf --height 40%)
```

Also, check out `--reverse` and `--layout` options if you prefer "top-down" layout instead of the default "bottom-up" layout.

```
vim $(fzf --height 40% --reverse)
```

You can add these options to `$FZF_DEFAULT_OPTS` so that they're applied by default. For example,

```
export FZF_DEFAULT_OPTS='--height 40% --layout=reverse --border'
```

## Search syntax

Unless otherwise specified, fzf starts in "extended-search mode" where you can type in multiple search terms delimited by spaces. e.g. `^music .mp3$ sbtrkt !fire`

Token	Match type	Description
<code>sbtrkt</code>	fuzzy-match	Items that match <code>sbtrkt</code>
<code>'wild</code>	exact-match (quoted)	Items that include <code>wild</code>
<code>^music</code>	prefix-exact-match	Items that start with <code>music</code>


Token	Match type	Description
<code>.mp3\$</code>	suffix-exact-match	Items that end with <code>.mp3</code>
<code>!fire</code>	inverse-exact-match	Items that do not include <code>fire</code>
<code>!^music</code>	inverse-prefix-exact-match	Items that do not start with <code>music</code>
<code>!.mp3\$</code>	inverse-suffix-exact-match	Items that do not end with <code>.mp3</code>

If you don't prefer fuzzy matching and do not wish to "quote" every word, start `fzf` with `-e` or `--exact` option. Note that when `--exact` is set, `'` -prefix "unquotes" the term.

A single bar character term acts as an OR operator. For example, the following query matches entries that start with `core` and end with either `go`, `rb`, or `py`.

```
^core go$ | rb$ | py$
```

## Environment variables

- `FZF_DEFAULT_COMMAND`
  - Default command to use when input is tty
  - e.g. `export FZF_DEFAULT_COMMAND='fd --type f'`
  -  This variable is not used by shell extensions due to the slight difference in requirements.  
  
(e.g. `CTRL-T` runs `$FZF_CTRL_T_COMMAND` instead, `vim **<tab>` runs `_fzf_compgen_path()`, and `cd **<tab>` runs `_fzf_compgen_dir()`)  
  
The available options are described later in this document.
- `FZF_DEFAULT_OPTS`
  - Default options
  - e.g. `export FZF_DEFAULT_OPTS="--layout=reverse --inline-info"`

## Options

See the man page ( `man fzf` ) for the full list of options.

## Demo

If you learn by watching videos, check out this screencast by [@samoshkin](#) to explore `fzf` features.



EPISODE #3

## fzf - command line fuzzy finder

Using "junegunn/fzf" to search for files or any list in a shell and Vim

Full screen (f)

0:03 / 13:40

## Examples

---

- [Wiki page of examples](#)
  - *Disclaimer: The examples on this page are maintained by the community and are not thoroughly tested*
- [Advanced fzf examples](#)

## fzf-tmux script

---

[fzf-tmux](#) is a bash script that opens fzf in a tmux pane.

```
# usage: fzf-tmux [LAYOUT OPTIONS] [--] [FZF OPTIONS]

# See available options
fzf-tmux --help

# select git branches in horizontal split below (15 lines)
git branch | fzf-tmux -d 15

# select multiple words in vertical split on the left (20% of screen width)
cat /usr/share/dict/words | fzf-tmux -l 20% --multi --reverse
```

It will still work even when you're not on tmux, silently ignoring `-[pu]d1r` options, so you can invariably use `fzf-tmux` in your scripts.

Alternatively, you can use `--height HEIGHT[%]` option not to start fzf in fullscreen mode.

fzf --height 40%

## Key bindings for command-line

---

The install script will setup the following key bindings for bash, zsh, and fish.

- CTRL-T - Paste the selected files and directories onto the command-line
  - Set `FZF_CTRL_T_COMMAND` to override the default command
  - Set `FZF_CTRL_T_OPTS` to pass additional options
- CTRL-R - Paste the selected command from history onto the command-line
  - If you want to see the commands in chronological order, press `CTRL-R` again which toggles sorting by relevance
  - Set `FZF_CTRL_R_OPTS` to pass additional options
- ALT-C - cd into the selected directory
  - Set `FZF_ALT_C_COMMAND` to override the default command
  - Set `FZF_ALT_C_OPTS` to pass additional options

If you're on a tmux session, you can start fzf in a tmux split-pane or in a tmux popup window by setting `FZF_TMUX_OPTS` (e.g. `-d 40%`). See `fzf-tmux --help` for available options.

More tips can be found on [the wiki page](#).

## Fuzzy completion for bash and zsh

---

### Files and directories

Fuzzy completion for files and directories can be triggered if the word before the cursor ends with the trigger sequence, which is by default `**`.

- `COMMAND [DIRECTORY/][FUZZY_PATTERN]**<TAB>`  
  
# Files under the current directory  
# - You can select multiple items with TAB key  
vim **\*\*<TAB>**  
  
# Files under parent directory  
vim **../\*\*<TAB>**  
  
# Files under parent directory that match `fzf`  
vim **../fzf\*\*<TAB>**  
  
# Files under your home directory  
vim **~/\*\*<TAB>**

```
# Directories under current directory (single-selection)
cd **<TAB>

# Directories under ~/github that match `fzf`
cd ~/github/fzf**<TAB>
```

## Process IDs

Fuzzy completion for PIDs is provided for kill command. In this case, there is no trigger sequence; just press the tab key after the kill command.

```
# Can select multiple processes with <TAB> or <Shift-TAB> keys
kill -9 <TAB>
```

## Host names

For ssh and telnet commands, fuzzy completion for hostnames is provided. The names are extracted from /etc/hosts and ~/.ssh/config.

```
ssh **<TAB>
telnet **<TAB>
```

## Environment variables / Aliases

```
unset **<TAB>
export **<TAB>
unalias **<TAB>
```

## Settings

```
# Use ~~ as the trigger sequence instead of the default **
export FZF_COMPLETION_TRIGGER='~~'

# Options to fzf command
export FZF_COMPLETION_OPTS='--border --info=inline'

# Use fd (https://github.com/sharkdp/fd) instead of the default find
# command for listing path candidates.
# - The first argument to the function ($1) is the base path to start traversal
# - See the source code (completion.{bash,zsh}) for the details.
_fzf_compgen_path() {
    fd --hidden --follow --exclude ".git" . "$1"
}
```

```

# Use fd to generate the list for directory completion
_fzf_compgen_dir() {
    fd --type d --hidden --follow --exclude ".git" . "$1"
}

# (EXPERIMENTAL) Advanced customization of fzf options via _fzf_comprun function
# - The first argument to the function is the name of the command.
# - You should make sure to pass the rest of the arguments to fzf.
_fzf_comprun() {
    local command=$1
    shift

    case "$command" in
        cd)          fzf "$@" --preview 'tree -C {} | head -200' ;;
        export|unset) fzf "$@" --preview "eval 'echo \${}'" ;;
        ssh)         fzf "$@" --preview 'dig {}' ;;
        *)           fzf "$@" ;;
    esac
}

```

## Supported commands

On bash, fuzzy completion is enabled only for a predefined set of commands ( `complete | grep _fzf` to see the list). But you can enable it for other commands as well by using `_fzf_setup_completion` helper function.

```

# usage: _fzf_setup_completion path|dir|var|alias|host COMMANDS...
_fzf_setup_completion path ag git kubect1
_fzf_setup_completion dir tree

```

## Custom fuzzy completion

*(Custom completion API is experimental and subject to change)*

For a command named "COMMAND", define `_fzf_complete_COMMAND` function using `_fzf_complete` helper.

```

# Custom fuzzy completion for "doge" command
# e.g. doge **<TAB>
_fzf_complete_doge() {
    _fzf_complete --multi --reverse --prompt="doge> " -- "$@" < <(
        echo very
        echo wow
        echo such
        echo doge
    )
}

```

- The arguments before `--` are the options to `fzf`.
- After `--`, simply pass the original completion arguments unchanged ( `"$@"` ).
- Then, write a set of commands that generates the completion candidates and feed its output to the function using process substitution ( `< <(...)` ).

`zsh` will automatically pick up the function using the naming convention but in `bash` you have to manually associate the function with the command using the `complete` command.

```
[ -n "$BASH" ] && complete -F _fzf_complete_doge -o default -o bashdefault doge
```

If you need to post-process the output from `fzf`, define `_fzf_complete_COMMAND_post` as follows.

```
_fzf_complete_foo() {
  _fzf_complete --multi --reverse --header-lines=3 -- "$@" < <(
    ls -al
  )
}
```

```
_fzf_complete_foo_post() {
  awk '{print $NF}'
}
```

```
[ -n "$BASH" ] && complete -F _fzf_complete_foo -o default -o bashdefault foo
```

## Vim plugin

---

See [README-VIM.md](#).

## Advanced topics

---

### Performance

`fzf` is fast and is [getting even faster](#). Performance should not be a problem in most use cases. However, you might want to be aware of the options that affect performance.

- `--ansi` tells `fzf` to extract and parse ANSI color codes in the input, and it makes the initial scanning slower. So it's not recommended that you add it to your `$FZF_DEFAULT_OPTS`.
- `--nth` makes `fzf` slower because it has to tokenize each line.
- `--with-nth` makes `fzf` slower as `fzf` has to tokenize and reassemble each line.
- If you absolutely need better performance, you can consider using `--algo=v1` (the default being `v2`) to make `fzf` use a faster greedy algorithm. However, this algorithm is

not guaranteed to find the optimal ordering of the matches and is not recommended.

## Executing external programs

You can set up key bindings for starting external processes without leaving `fzf` ( `execute` , `execute-silent` ).

```
# Press F1 to open the file with less without leaving fzf
# Press CTRL-Y to copy the line to clipboard and aborts fzf (requires pbcopy)
fzf --bind 'f1:execute(less -f {}),ctrl-y:execute-silent(echo {} | pbcopy)+abort'
```

See *KEY BINDINGS* section of the man page for details.

## Reloading the candidate list

By binding `reload` action to a key or an event, you can make `fzf` dynamically reload the candidate list. See [#1750](#) for more details.

### 1. Update the list of processes by pressing CTRL-R

```
FZF_DEFAULT_COMMAND='ps -ef' \
fzf --bind 'ctrl-r:reload($FZF_DEFAULT_COMMAND)' \
    --header 'Press CTRL-R to reload' --header-lines=1 \
    --height=50% --layout=reverse
```

### 2. Switch between sources by pressing CTRL-D or CTRL-F

```
FZF_DEFAULT_COMMAND='find . -type f' \
fzf --bind 'ctrl-d:reload(find . -type d),ctrl-f:reload($FZF_DEFAULT_COMMAND)' \
    --height=50% --layout=reverse
```

### 3. Interactive ripgrep integration

The following example uses `fzf` as the selector interface for `ripgrep`. We bound `reload` action to `change` event, so every time you type on `fzf`, the `ripgrep` process will restart with the updated query string denoted by the placeholder expression `{q}`. Also, note that we used `--disabled` option so that `fzf` doesn't perform any secondary filtering.

```
INITIAL_QUERY=""
RG_PREFIX="rg --column --line-number --no-heading --color=always --smart-case "
FZF_DEFAULT_COMMAND="$RG_PREFIX '$INITIAL_QUERY'" \
fzf --bind "change:reload:$RG_PREFIX {q} || true" \
    --ansi --disabled --query "$INITIAL_QUERY" \
    --height=50% --layout=reverse
```



If ripgrep doesn't find any matches, it will exit with a non-zero exit status, and fzf will warn you about it. To suppress the warning message, we added `|| true` to the command, so that it always exits with 0.

See ["Using fzf as interactive Ripgrep launcher"](#) for a fuller example with preview window options.

## Preview window

When the `--preview` option is set, fzf automatically starts an external process with the current line as the argument and shows the result in the split window. Your `$SHELL` is used to execute the command with `$SHELL -c COMMAND`. The window can be scrolled using the mouse or custom key bindings.

```
# {} is replaced with the single-quoted string of the focused line
fzf --preview 'cat {}'
```

Preview window supports ANSI colors, so you can use any program that syntax-highlights the content of a file, such as [Bat](#) or [Highlight](#):

```
fzf --preview 'bat --style=numbers --color=always --line-range :500 {}'
```

You can customize the size, position, and border of the preview window using `--preview-window` option, and the foreground and background color of it with `--color` option. For example,

```
fzf --height 40% --layout reverse --info inline --border \
    --preview 'file {}' --preview-window up,1,border-horizontal \
    --color 'fg:#bbccdd,fg+:#ddeeff,bg:#334455,preview-bg:#223344,border:#778899'
```

See the man page ( `man fzf` ) for the full list of options.

For more advanced examples, see [Key bindings for git with fzf \(code\)](#).

Since fzf is a general-purpose text filter rather than a file finder, **it is not a good idea to add `--preview` option to your `$FZF_DEFAULT_OPTS`**.

```
# *****
# ** DO NOT DO THIS! **
# *****
export FZF_DEFAULT_OPTS='--preview "bat --style=numbers --color=always --line-range :
# bat doesn't work with any input other than the list of files'
```

```
ps -ef | fzf
seq 100 | fzf
history | fzf
```

## Tips

---

### Respecting .gitignore

You can use [fd](#), [ripgrep](#), or [the silver searcher](#) instead of the default find command to traverse the file system while respecting `.gitignore`.

```
# Feed the output of fd into fzf
fd --type f | fzf

# Setting fd as the default source for fzf
export FZF_DEFAULT_COMMAND='fd --type f'

# Now fzf (w/o pipe) will use fd instead of find
fzf

# To apply the command to CTRL-T as well
export FZF_CTRL_T_COMMAND="$FZF_DEFAULT_COMMAND"
```

If you want the command to follow symbolic links and don't want it to exclude hidden files, use the following command:

```
export FZF_DEFAULT_COMMAND='fd --type f --hidden --follow --exclude .git'
```

### Fish shell

CTRL-T key binding of fish, unlike those of bash and zsh, will use the last token on the command-line as the root directory for the recursive search. For instance, hitting CTRL-T at the end of the following command-line

```
ls /var/
```

will list all files and directories under `/var/`.

When using a custom `FZF_CTRL_T_COMMAND`, use the unexpanded `$dir` variable to make use of this feature. `$dir` defaults to `.` when the last token is not a valid directory. Example:

```
set -g FZF_CTRL_T_COMMAND "command find -L \"$dir\" -type f 2> /dev/null | sed '1d; s#^\\" data-bbox="86 954 911 966">
```

## Related projects

<https://github.com/junegunn/fzf/wiki/Related-projects>

## License

The MIT License (MIT)

Copyright (c) 2013-2021 Junegunn Choi

### Releases 15

 0.30.0 Latest  
on 4 Apr

[+ 14 releases](#)

### Sponsor this project

 <https://paypal.me/junegunn>

 <https://www.buymeacoffee.com/junegunn>

### Packages

No packages published

### Used by 17



### Contributors 202



[+ 191 contributors](#)

### Languages

● Go 69.5%   ● Ruby 15.1%   ● Shell 9.0%   ● Vim Script 5.2%   ● Makefile 0.8%   ● PowerShell 0.3%

● Dockerfile 0.1%