Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'fish-completions.1'

$ man fish-completions.1

FISH-COMPLETIONS(1) fish-shell FISH-COMPLETIONS(1)

To specify a completion, use the complete command. complete takes as a parameter the name of the command to specify a completion for. For example, to add a completion for the program myprog, start the completion command with complete -c myprog ...

For a complete description of the various switches accepted by the complete command, see the docu? mentation for the complete <> builtin, or write complete --help inside the fish shell.

To provide a list of possible completions for myprog, use the -a switch. If myprog accepts the ar? guments start and stop, this can be specified as complete -c myprog -a 'start stop'. The argument to the -a switch is always a single string. At completion time, it will be tokenized on spaces and tabs, and variable expansion, command substitution and other forms of parameter expansion will take place:

```
# If myprog can list the valid outputs with the list-outputs subcommand:
complete -c myprog -l output -a '(myprog list-outputs)'
```

fish has a special syntax to support specifying switches accepted by a command. The switches -s, -l and -o are used to specify a short switch (single character, such as -l), a gnu style long switch (such as --color) and an old-style long switch (with one -, like -shuffle), re? spectively. If the command 'myprog' has an option that can be written as -o or --output, that is:

```
complete -c myprog -s o -l output
```

If this option takes an optional argument, you would also add --argument or -a, and give that the possible arguments:

```
complete -c myprog -s o -l output -a "yes no"
```

This offers the arguments "yes" and "no" for:

```
> myprog -o<TAB>
> myprog --output=<TAB>
```

By default, option arguments are optional, so the candidates are only offered directly at?
tached like that, so they aren't given in this case:

```
> myprog -o <TAB>
```

Usually options require a parameter, so you would give --require-parameter / -r:

```
complete -c myprog -s o -l output -ra "yes no"
```

which offers yes/no in these cases:

```
> myprog -o<TAB>
> myprog --output=<TAB>
> myprog -o <TAB>
> myprog --output <TAB>
```

Fish will also offer files by default, in addition to the arguments you specified. You would
either inhibit file completion for a single option:

```
complete -c myprog -s o -l output --no-files -ra "yes no"
```

or with a specific condition:

```
complete -c myprog -f --condition '__fish_seen_subcommand_from somesubcommand'
```

or you can disable file completions globally for the command:

```
complete -c myprog -f
```

If you have disabled them globally, you can enable them just for a specific condition or op?
tion with the --force-files / -F option:

```
# Disable files by default
complete -c myprog -f
# but reenable them for --config-file
complete -c myprog -l config-file --force-files -r
```

As a more comprehensive example, here's a commented excerpt of the completions for systemd's
timedatectl:

```
# All subcommands that timedatectl knows - this is useful for later.
set -l commands status set-time set-timezone list-timezones set-local-rtc set-ntp
# Disable file completions for the entire command
# because it does not take files anywhere
# Note that this can be undone by using "-F".
#
```

```
# File completions also need to be disabled
# if you want to have more control over what files are offered
# (e.g. just directories, or just files ending in ".mp3").
complete -c timedatectl -f
# This line offers the subcommands
# -"status",
# -"set-timezone",
# -"set-time"
# -"list-timezones"
# if no subcommand has been given so far.
#
# The `-n`/`--condition` option takes script as a string, which it executes.
# If it returns true, the completion is offered.
# Here the condition is the `__fish_seen_subcommand_from` helper function.
# It returns true if any of the given commands is used on the commandline,
# as determined by a simple heuristic.
# For more complex uses, you can write your own function.
# See e.g. the git completions for an example.
#
complete -c timedatectl -n "not __fish_seen_subcommand_from $commands" \
    -a "status set-time set-timezone list-timezones"
# If the "set-timezone" subcommand is used,
# offer the output of `timedatectl list-timezones` as completions.
# Each line of output is used as a separate candidate,
# and anything after a tab is taken as the description.
# It's often useful to transform command output with `string` into that form.
complete -c timedatectl -n "__fish_seen_subcommand_from set-timezone" \
    -a "(timedatectl list-timezones)"
# Completion candidates can also be described via `-d`,
# which is useful if the description is constant.
# Try to keep these short, because that means the user gets to see more at once.
complete -c timedatectl -n "not __fish_seen_subcommand_from $commands" \
    -a "set-local-rtc" -d "Maintain RTC in local time"
```

```
# We can also limit options to certain subcommands by using conditions.
complete -c timedatectl -n "__fish_seen_subcommand_from set-local-rtc" \
    -l adjust-system-clock -d 'Synchronize system clock from the RTC'
# These are simple options that can be used everywhere.
complete -c timedatectl -s h -l help -d 'Print a short help text and exit'
complete -c timedatectl -l version -d 'Print a short version string and exit'
complete -c timedatectl -l no-pager -d 'Do not pipe output into a pager'
```

For examples of how to write your own complex completions, study the completions in /usr/share/fish/completions. (The exact path depends on your chosen installation prefix and may be slightly different)

USEFUL FUNCTIONS FOR WRITING COMPLETIONS

fish ships with several functions that may be useful when writing command-specific comple? tions. Most of these function names begin with the string __fish_. Such functions are inter? nal to fish and their name and interface may change in future fish versions. A few of these functions are described here.

Functions beginning with the string __fish_print_ print a newline separated list of strings. For example, __fish_print_filesystems prints a list of all known file systems. Functions be? ginning with __fish_complete_ print out a newline separated list of completions with de? scriptions. The description is separated from the completion by a tab character.

? __fish_complete_directories STRING DESCRIPTION performs path completion on STRING, allow? ing only directories, and giving them the description DESCRIPTION.

? __fish_complete_path STRING DESCRIPTION performs path completion on STRING, giving them the description DESCRIPTION.

? __fish_complete_groups prints a list of all user groups with the groups members as de? scription.

? __fish_complete_pids prints a list of all processes IDs with the command name as descrip? tion.

? __fish_complete_suffix SUFFIX performs file completion but sorts files ending in SUFFIX first. This is useful in conjunction with complete --keep-order.

? __fish_complete_users prints a list of all users with their full name as description.

? __fish_print_filesystems prints a list of all known file systems. Currently, this is a static list, and not dependent on what file systems the host operating system actually un? derstands.

? __fish_print_hostnames prints a list of all known hostnames. This function searches the fstab for nfs servers, ssh for known hosts and checks the /etc/hosts file.

? __fish_print_interfaces prints a list of all known network interfaces.

WHERE TO PUT COMPLETIONS

Completions can be defined on the commandline or in a configuration file, but they can also be automatically loaded. Fish automatically searches through any directories in the list variable $fish_complete_path, and any completions defined are automatically loaded when needed. A completion file must have a filename consisting of the name of the command to com? plete and the suffix .fish.

By default, Fish searches the following for completions, using the first available file that it finds:

? A directory for end-users to keep their own completions, usually ~/.config/fish/comple? tions (controlled by the XDG_CONFIG_HOME environment variable);

? A directory for systems administrators to install completions for all users on the system, usually /etc/fish/completions;

? A user-specified directory for third-party vendor completions, usually ~/.lo? cal/share/fish/vendor_completions.d (controlled by the XDG_DATA_HOME environment vari? able);

? A directory for third-party software vendors to ship their own completions for their soft? ware, usually /usr/share/fish/vendor_completions.d;

? The completions shipped with fish, usually installed in /usr/share/fish/completions; and

? Completions automatically generated from the operating system's manual, usually stored in ~/.cache/fish/generated_completions (controlled by XDG_CACHE_HOME environment variable).

These paths are controlled by parameters set at build, install, or run time, and may vary from the defaults listed above.

This wide search may be confusing. If you are unsure, your completions probably belong in ~/.config/fish/completions.

If you have written new completions for a common Unix command, please consider sharing your work by submitting it via the instructions in Further help and development <#more-help>.

If you are developing another program and would like to ship completions with your program, install them to the "vendor" completions directory. As this path may vary from system to system, the pkgconfig framework should be used to discover this path with the output of pkg-config --variable completionsdir fish.

Author

fish-shell developers