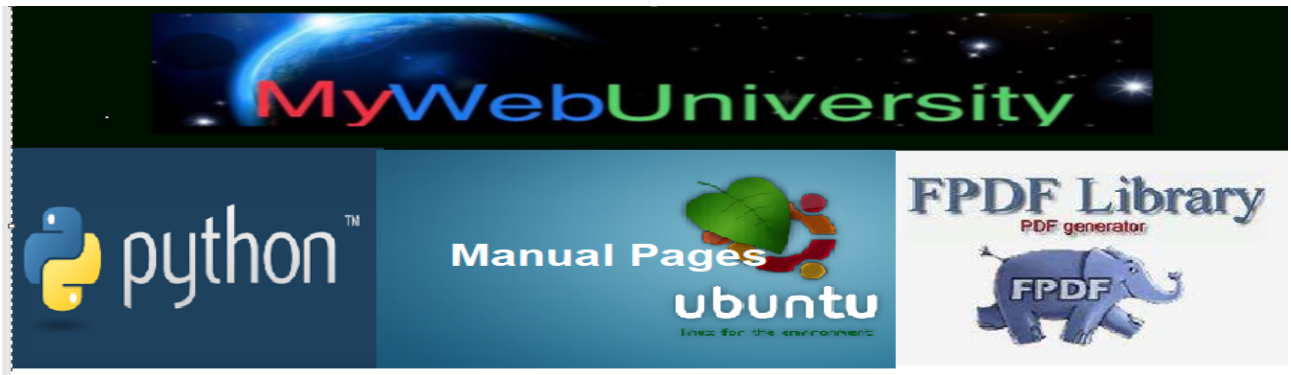

Full credit is given to the above companies including the OS that this PDF file was generated!

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

$ man fish-faq.1

FISH-FAQ(1) fish-shell FISH-FAQ(1)

WHAT IS THE EQUIVALENT TO THIS THING FROM BASH (OR OTHER SHELLS)?

See Fish for bash users <>

HOW DO I SET OR CLEAR AN ENVIRONMENT VARIABLE?

Use the set <> command:

set -x key value # typically set -gx key value

set -e key

Since fish 3.1 you can set an environment variable for one command using the key=value some command syntax, like in other shells. The two lines below behave identically - unlike other shells, fish will output value both times:

key=value echo $key

begin; set -lx key value; echo $key; end

Note that "exported" is not a scope <#variables-scope>, but an additional bit of state. A variable can be global and exported or local and exported or even universal and exported. Typically it makes sense to make an exported variable global.

HOW DO I CHECK WHETHER A VARIABLE IS DEFINED?

Use set -q var. For example, if set -q var; echo variable defined; end. To check multiple variables you can combine with and and or like so:

```
if set -q var1; or set -q var2
    echo either variable defined
end
```

Keep in mind that a defined variable could also be empty, either by having no elements (if set like set var) or only empty elements (if set like set var ""). Read on for how to deal

with those.

HOW DO I CHECK WHETHER A VARIABLE IS NOT EMPTY?

Use string length -q -- $var. For example, if string length -q -- $var; echo not empty; end. Note that string length will interpret a list of multiple variables as a disjunction (meaning any/or):

```
if string length -q -- $var1 $var2 $var3
    echo at least one of these variables is not empty
end
```

Alternatively, use test -n "$var", but remember that the variable must be double-quoted. For example, if test -n "$var"; echo not empty; end. The test command provides its own and (-a) and or (-o):

```
if test -n "$var1" -o -n "$var2" -o -n "$var3"
    echo at least one of these variables is not empty
end
```

If you want to know if a variable has no elements, use set -q var[1].

WHY DOESN'T SET -UX (EXPORTED UNIVERSAL VARIABLES) SEEM TO WORK?

A global variable of the same name already exists.

Environment variables such as EDITOR or TZ can be set universally using set -Ux. However, if there is an environment variable already set before fish starts (such as by login scripts or system administrators), it is imported into fish as a global variable. The variable scopes <#variables-scope> are searched from the "inside out", which means that local vari? ables are checked first, followed by global variables, and finally universal variables. This means that the global value takes precedence over the universal value.

To avoid this problem, consider changing the setting which fish inherits. If this is not possible, add a statement to your configuration file <#configuration> (usually ~/.con? fig/fish/config.fish):

```
set -gx EDITOR vim
```

HOW DO I RUN A COMMAND EVERY LOGIN? WHAT'S FISH'S EQUIVALENT TO .BASHRC OR .PROFILE?

Edit the file ~/.config/fish/config.fish [1], creating it if it does not exist (Note the leading period).

Unlike .bashrc and .profile, this file is always read, even in non-interactive or login shells.

To do something only in interactive shells, check status is-interactive like:

```
if status is-interactive
    # use the coolbeans theme
    fish_config theme choose coolbeans
end
```

[1] The "~/.config" part of this can be set via $XDG_CONFIG_HOME, that's just the default.

HOW DO I SET MY PROMPT?

The prompt is the output of the fish_prompt function. Put it in ~/.config/fish/functions/fish_prompt.fish. For example, a simple prompt is:

```
function fish_prompt
    set_color $fish_color_cwd
    echo -n (prompt_pwd)
    set_color normal
    echo -n ' > '
end
```

You can also use the Web configuration tool, fish_config <>, to preview and choose from a gallery of sample prompts.

Or you can use fish_config from the commandline:

```
> fish_config prompt show
# displays all the prompts fish ships with
> fish_config prompt choose disco
# loads the disco prompt in the current shell
> fish_config prompt save
# makes the change permanent
```

If you want to modify your existing prompt, you can use funced <> and funcsave <> like:

```
>_ funced fish_prompt
# This opens up your editor (set in $EDITOR).
# Modify the function,
# save the file and repeat to your liking.
# Once you are happy with it:
>_ funcsave fish_prompt
```

This also applies to fish_right_prompt <> and fish_mode_prompt <>.

WHY DOES MY PROMPT SHOW A [I]?

That's the fish_mode_prompt <>. It is displayed by default when you've activated vi mode us?

ing fish_vi_key_bindings.

If you haven't activated vi mode on purpose, you might have installed a third-party theme or plugin that does it.

If you want to change or disable this display, modify the fish_mode_prompt function, for instance via funced <>.

HOW DO I CUSTOMIZE MY SYNTAX HIGHLIGHTING COLORS?

Use the web configuration tool, fish_config <>, or alter the fish_color family of environment variables <#variables-color>.

You can also use fish_config on the commandline, like:

```
> fish_config theme show
# to demonstrate all the colorschemes
> fish_config theme choose coolbeans
# to load the "coolbeans" theme
> fish_config theme save
# to make the change permanent
```

HOW DO I CHANGE THE GREETING MESSAGE?

Change the value of the variable fish_greeting or create a fish_greeting <> function. For example, to remove the greeting use:

```
set -U fish_greeting
```

Or if you prefer not to use a universal variable, use:

```
set -g fish_greeting
```

in config.fish <#configuration>.

HOW DO I RUN A COMMAND FROM HISTORY?

Type some part of the command, and then hit the up (?) or down (?) arrow keys to navigate through history matches, or press ctrl-r to open the history in a searchable pager. In this pager you can press ctrl-r or ctrl-s to move to older or younger history respectively.

Additional default key bindings include ctrl-p (up) and ctrl-n (down). See Searchable command history <#history-search> for more information.

WHY DOESN'T HISTORY SUBSTITUTION ("!$" ETC.) WORK?

Because history substitution is an awkward interface that was invented before interactive line editing was even possible. Instead of adding this pseudo-syntax, fish opts for nice history searching and recall features. Switching requires a small change of habits: if you want to modify an old line/word, first recall it, then edit.

As a special case, most of the time history substitution is used as sudo !!. In that case press alt-s, and it will recall your last commandline with sudo prefixed (or toggle a sudo prefix on the current commandline if there is anything).

In general, fish's history recall works like this:

? Like other shells, the Up arrow, up recalls whole lines, starting from the last executed line. So instead of typing !!, you would hit the up-arrow.

? If the line you want is far back in the history, type any part of the line and then press Up one or more times. This will filter the recalled lines to ones that include this text, and you will get to the line you want much faster. This replaces "!vi", "!?bar.c" and the like. If you want to see more context, you can press ctrl-r to open the history in the pager.

? alt-up recalls individual arguments, starting from the last argument in the last executed line. This can be used instead of "!$".

See documentation <#editor> for more details about line editing in fish.

That being said, you can use Abbreviations <#abbreviations> to implement history substitu?tion. Here's !! only:

```
function last_history_item; echo $history[1]; end
abbr -a !! --position anywhere --function last_history_item
```

Run this and !! will be replaced with the last history entry, anywhere on the commandline.

Put it into config.fish <#configuration> to keep it.

HOW DO I RUN A SUBCOMMAND? THE BACKTICK DOESN'T WORK!

fish uses parentheses for subcommands. For example:

```
for i in (ls)
    echo $i
end
```

It also supports the familiar $() syntax, even in quotes. Backticks are not supported be?cause they are discouraged even in POSIX shells. They nest poorly and are hard to tell from single quotes (').

MY COMMAND (PKG-CONFIG) GIVES ITS OUTPUT AS A SINGLE LONG STRING?

Unlike other shells, fish splits command substitutions only on newlines, not spaces or tabs or the characters in $IFS.

That means if you run

```
count (printf '%s ' a b c)
```

It will print 1, because the "a b c " is used in one piece. But if you do

```
count (printf '%s\n' a b c)
```

it will print 3, because it gave count the arguments "a", "b" and "c" separately.

In the overwhelming majority of cases, splitting on spaces is unwanted, so this is an improvement. This is why you hear about problems with filenames with spaces, after all.

However sometimes, especially with pkg-config and related tools, splitting on spaces is needed.

In these cases use string split -n " " like:

```
g++ example_01.cpp (pkg-config --cflags --libs gtk+-2.0 | string split -n " ")
```

The -n is so empty elements are removed like POSIX shells would do.

HOW DO I GET THE EXIT STATUS OF A COMMAND?

Use the $status variable. This replaces the $? variable used in other shells.

```
somecommand
if test $status -eq 7
    echo "That's my lucky number!"
end
```

If you are only interested in success or failure, you can run the command directly as the if-condition:

```
if somecommand
    echo "Command succeeded"
else
    echo "Command failed"
end
```

Or if you just want to do one command in case the first succeeded or failed, use and or or:

```
somecommand
or someothercommand
```

See the Conditions <#syntax-conditional> and the documentation for test <> and if <> for more information.

MY COMMAND PRINTS NO MATCHES FOR WILDCARD BUT WORKS IN BASH

In short: quote <#quotes> or escape <#escapes> the wildcard:

```
scp user@ip:/dir/"string-*"
```

When fish sees an unquoted *, it performs wildcard expansion <#expand-wildcard>. That means it tries to match filenames to the given string.

If the wildcard doesn't match any files, fish prints an error instead of running the com?
mand:

```
> echo *this*does*not*exist
fish: No matches for wildcard '*this*does*not*exist'. See `help expand`.
echo *this*does*not*exist
     ^
```

Now, bash also tries to match files in this case, but when it doesn't find a match, it passes along the literal wildcard string instead.

That means that commands like the above

```
scp user@ip:/dir/string-*
```

or

```
apt install postgres-*
```

appear to work, because most of the time the string doesn't match and so it passes along the string-*, which is then interpreted by the receiving program.

But it also means that these commands can stop working at any moment once a matching file is encountered (because it has been created or the command is executed in a different working directory), and to deal with that bash needs workarounds like

```
for f in ./*.mpg; do
    # We need to test if the file really exists because
    # the wildcard might have failed to match.
    test -f "$f" || continue
    mympgviewer "$f"
done
```

(from <http://mywiki.wooledge.org/BashFAQ/004>)

For these reasons, fish does not do this, and instead expects asterisks to be quoted or es? caped if they aren't supposed to be expanded.

This is similar to bash's "failglob" option.

WHY WON'T SSH/SCP/RSYNC CONNECT PROPERLY WHEN FISH IS MY LOGIN SHELL?

This problem may show up as messages like "Received message too long", "open terminal failed: not a terminal", "Bad packet length", or "Connection refused" with strange output in ssh_exchange_identification messages in the debug log.

This usually happens because fish reads the user configuration file <#configuration> (~/.config/fish/config.fish) always, whether it's in an interactive or login or non-interac?

tive or non-login shell.

This simplifies matters, but it also means when config.fish generates output, it will do that even in non-interactive shells like the one ssh/scp/rsync start when they connect.

Anything in config.fish that produces output should be guarded with status is-interactive (or status is-login if you prefer):

```
if status is-interactive
  ...
end
```

The same applies for example when you start tmux in config.fish without guards, which will cause a message like sessions should be nested with care, unset $TMUX to force.

I'M GETTING WEIRD GRAPHICAL GLITCHES (A STAIRCASE EFFECT, GHOST CHARACTERS, CURSOR IN THE WRONG PO?

SITION,...)?

In a terminal, the application running inside it and the terminal itself need to agree on the width of characters in order to handle cursor movement.

This is more important to fish than other shells because features like syntax highlighting and autosuggestions are implemented by moving the cursor.

Sometimes, there is disagreement on the width. There are numerous causes and fixes for this:

? It is possible the character is too new for your system to know - in this case you need to refrain from using it.

? Fish or your terminal might not know about the character or handle it wrong - in this case fish or your terminal needs to be fixed, or you need to update to a fixed version.

? The character has an "ambiguous" width and fish thinks that means a width of X while your terminal thinks it's Y. In this case you either need to change your terminal's configura? tion or set $fish_ambiguous_width to the correct value.

? The character is an emoji and the host system only supports Unicode 8, while you are run? ning the terminal on a system that uses Unicode >= 9. In this case set $fish_emoji_width to 2.

This also means that a few things are unsupportable:

? Non-monospace fonts - there is no way for fish to figure out what width a specific charac? ter has as it has no influence on the terminal's font rendering.

? Different widths for multiple ambiguous width characters - there is no way for fish to know which width you assign to each character.

FISH DOES NOT WORK IN A SPECIFIC TERMINAL

The terminal might not meet all of fish's requirements <>. Please report this to your ter?

minal's and to fish's issue tracker.

UNINSTALLING FISH

If you want to uninstall fish, first make sure fish is not set as your shell. Run chsh -s

/bin/bash if you are not sure.

If you installed it with a package manager, use that package manager's uninstall function.

If you built fish yourself, assuming you installed it to /usr/local, do this:

```
rm -Rf /usr/local/etc/fish /usr/local/share/fish ~/.config/fish
rm /usr/local/share/man/man1/fish*.1
cd /usr/local/bin
rm -f fish fish_indent
```

Author

fish-shell developers

Copyright

fish-shell developers

4.2 Jan 02, 2026 FISH-FAQ(1)