Fish Shell Cheatsheet

A comprehensive cheat sheet covering essential Fish shell commands, syntax, and configurations for improved productivity and customization.

Basics & Navigation

Shell Navigation

`cd directory_name`	Change directory to `directory_name`.
`cd ..`	Move one directory up.
`cd -`	Return to the previous directory.
`pwd`	Print the current working directory.
`ls`	List files and directories in the current directory.
`ls -l`	List files with detailed information (permissions, size, etc.).
`ls -a`	List all files, including hidden files.

Basic Commands

`echo 'text'`	Prints the specified text to the terminal.
`printf '%s\n' 'text'`	Prints the specified text with a trailing newline.
`read my_variable`	Reads input from stdin and assigns it to `my_variable`.
`clear`	Clears the terminal screen.
`history`	Displays the command history.

Keybindings

`Ctrl+A`	Move cursor to the beginning of the line.
`Ctrl+E`	Move cursor to the end of the line.
`Alt+Left`	Jump to the previous word.
`Alt+Right`	Jump to the next word.
`Up/Down Arrows`	Switch to the previous/next command in history.
`Alt+Up/Down`	Switch to the previous/next arguments.
`Ctrl+U`	Delete from cursor to the beginning of the line.
`Ctrl+C`	Cancel the current command.

Variables & Loops

Variable Management

`set my_variable 'value'`	Sets a variable named `my_variable` to 'value'.
`set -g my_variable 'value'`	Sets a global variable accessible in all scopes.
`set -l my_variable 'value'`	Sets a local variable, only accessible within the current scope.
`set --erase my_variable`	Removes the variable `my_variable`.
`echo $my_variable`	Prints the value of `my_variable`.
`echo $my_variable[1..3]`	Prints a slice of the variable (characters 1 to 3).

Loop Structures

`for i in item1 item2 item3; ...; end`	Iterates over a list of items.
`for i in (seq 1 5); ...; end`	Iterates over a sequence of numbers from 1 to 5.
`while condition; ...; end`	Executes a block of code as long as the condition is true.
`break`	Exits the current loop.
`continue`	Skips the current iteration and continues with the next.

Arithmetic Operations

`math 1 + 2`	Performs addition.
`math 5 - 3`	Performs subtraction.
`math 4 \* 6`	Performs multiplication.
`math 8 / 2`	Performs division.
`math 7 % 3`	Calculates the modulo (remainder).
`math 2 ^ 3`	Performs exponentiation.
`set counter (math $counter + 1)`	Increments a variable `counter` by 1.

Conditionals & Strings

Conditional Statements

`if condition; ...; end`	Executes a block of code if the condition is true.
`else`	Executes a block of code if the previous `if` condition is false.
`else if condition; ...; end`	Chains another condition to check if the initial `if` is false.
`test condition`	Evaluates a condition; returns 0 if true, 1 if false.
`and`	Logical AND operator for combining conditions.
`or`	Logical OR operator for combining conditions.
`not`	Logical NOT operator to negate a condition.

String Manipulation

`string length 'text'`	Returns the length of the string 'text'.
`string sub --start 2 --length 3 'example'`	Extracts a substring of length 3 starting from index 2.
`string match --regex 'pattern' 'string'`	Matches a string against a regular expression; returns the match or nothing.
`string replace --all 'old' 'new' 'string'`	Replaces all occurrences of 'old' with 'new' in 'string'.
`string join 'separator' item1 item2 ...`	Joins items together with the specified separator.
`string split 'separator' 'string'`	Splits a string into an array based on the separator.

String Matching Patterns

`x?`	Matches zero or one occurrences of `x`.
`x*`	Matches zero or more occurrences of `x`.
`x+`	Matches one or more occurrences of `x`.
`x{n}`	Matches exactly `n` occurrences of `x`.
`x{n,m}`	Matches between `n` and `m` occurrences of `x`.
`[xy]`	Matches either `x` or `y`.
`[^xy]`	Matches any character that is not `x` or `y`.

Functions & Events

Function Definition

`function my_function; ...; end`	Defines a new function named `my_function`.
`function my_function -a argument1 argument2; ...; end`	Defines a function accepting specific arguments.
`function my_function -d 'description'; ...; end`	Adds a description to the function.
`functions --erase my_function`	Removes the function `my_function`.
`return`	Exits the function, optionally returning a value.
`$argv`	Array containing the arguments passed to the function.

Event Handling

`emit my_event`	Emits an event named `my_event`.
`function hook_function --on-event my_event; ...; end`	Defines a hook function that runs when `my_event` is emitted.
`function hook_function --on-variable my_variable; ...; end`	Defines a hook function that runs when `my_variable` is changed.
`function hook_function --on-signal SIGINT; ...; end`	Defines a hook function that runs when the `SIGINT` signal is received (e.g., Ctrl+C).

Process Communication

`> file`	Redirects the output to a file, overwriting its contents.
`>> file`	Appends the output to a file.
`| command`	Pipes the output to another command.
`(command)`	Command substitution; replaces the command with its output.
`(command | psub)`	Process substitution; creates a temporary file with the output of the command.

Abbreviations & Completions

Abbreviations

`abbr --add short_cmd 'long command'`	Adds an abbreviation; typing `short_cmd` will execute `long command`.
`abbr --erase short_cmd`	Removes the abbreviation `short_cmd`.
`abbr`	Lists all defined abbreviations.
Example: `abbr --add gcm 'git commit -m'`	Creates an abbreviation for `git commit -m`. Use: `gcm 'Your commit message'`

Command Completions

`complete --command cmd --arguments arg1 arg2`	Adds completions for the command `cmd` with specified arguments.
`complete --command cmd --erase`	Removes all completions for the command `cmd`.
`complete --command cmd --no-files`	Disables file completion for command `cmd`.
`complete --command cmd --force-files`	Forces file completion for command `cmd`.
`complete --command cmd --condition condition`	Adds completions based on a specified condition.
`complete --command cmd --description 'text'`	Adds a description for the completion.

Useful Built-in Functions

`__fish_seen_argument`	Checks whether a specific argument has been used.
`__fish_seen_subcommand_from`	Checks if a specific subcommand has been used.
`__fish_use_subcommand`	Checks if any subcommand is used.
`__fish_complete_directories`	Completes directories.
`__fish_complete_suffix`	Completes files with a specific suffix.
`__fish_complete_users`	Lists all users for completion.