Syntax

Variables

Three kinds: universal, global, and local. Universal variables are shared among all sessions with the same user on the same computer. Global variables are available in every scope in a single shell session. Local variables are block scoped.

Set a variable as universal with `-U`, as global with `-g`, or local with `-l`.

Declaring Variables

Command	Scope
`set -U`	Universal
`set -g`	Global
`set -l`	Local

Setting Variables

1. If you specify the scope in the set command, that scope is honored. If a variable with the same name exists in a different scope, that variable isn't changed.
2. If the scope is unspecified but a variable with the same name is already defined, that variable's scope is used.
3. If the scope is unspecified and has never been defined, the variable will be local to the current function (this is function scope, not block scope like -l). If no function is executing, the variable will be global.

Exporting Variables

Export a variable to the environment using `set -x`.

Arrays

Store multiple strings in one variable with an array.

Access an index

```
echo $PATH[3]
```

Iterate

```
for i in $PATH
    echo $i in the path
end
```

Built-in Variables

`argv`	array of arguments to a shell function
`history`	array containing the command history
`HOME`	the user's home directory
`PWD`	the current working directory
`status`	the exit status of the last foreground job to exit
`USER`	the current username
`PATH`	a global variable automatically reset in each new fish session

IO Redirection and Piping

Redirect stdin	`N<SOURCE_FILE` (N is optional, default is 0)
Redirect stdout	`N>DESTINATION` (N is optional; default is 1)
Redirect stderr	`N^DESTINATION` (N is optional; default is 2)
Redirect with appending	`>>` or `^^` + `DESTINATION_FILE`
Close FD	use `-` as `SOURCE_FILE` or `DESTINATION`
Pipe stdout	`command1 | command2`
Pipe a different FD	`command1 N>| command2`

Expansion

Support for Expansion in Quotes

Type	Var Exp?	Esc. Char
none	Yes	All
" "	Yes	\", \$, and \\
' '	No	\', \\

Command Expansion

Surround command in parentheses. If it returns multiple lines, they'll be joined with spaces.

Definition

Make an array called smurf containing “blue” and “small”:

```
set smurf blue small
```

Delete an element

```
set -e smurf[1]
```

Functions

Define a function like so:

```
function ll
    ls -l $argv
end
```

Access arguments using `$argv`, call the function using `ll`.

Jobs

When you execute a command, it starts a job. You can put a job in the background by adding the `&` suffix. You can suspend a currently running job using `Ctrl-Z`. You can put the suspended job in the background with `bg`. Finally, you can list all running jobs with `jobs`.

Chaining Commands

Each command ends in either a newline or a semicolon. Chain commands using `command1; and command2` or `command1; or command2`. `and` and `or` check the previous command’s exit status and act accordingly.

Aliases

To define an alias, either make a function or use `alias NAME DEFINITION`, which actually just defines a function for you.

Parameter Expansion

Fish supports more limited globbing than other shells; use `find` with command expansion for more complicated globs. Files beginning with `.` are ignored unless `.` is the first character in the glob.

Char	Behavior	Exception
?	any single character	/
*	any string of characters	/
**	any string of characters	none

Brace Expansion

Same as in bash.

```
echo input.c,h,txt
>> input.c input.h input.txt
```

Variable Expansion

A $ followed by a string of characters is expanded to the value of the environmental variable with that name. Surround with braces to separate from text.

Process Expansion

; % followed by a string is expanded into a PID according these rules:

1. If the string is `self`, insert the shell PID
2. If the string is the ID of a job, insert the process group ID for the job
3. If any child processes match the string, insert their PIDs
4. If any processes owned by the user match the string, insert their PIDs
5. else produce an error

Index Range Expansion

Select a range of values from an array using `..`:

```
echo (seq 10)[2..5 1..3]
>> 2 3 4 5 1 2 3
```

Editor Shortcuts

Complete current token	`Tab`
Accept autosuggestion	`at EOL: End/Ctrl-E/Right/Ctrl-F`
Move to BOL	`Home/Ctrl-A`
Move to EOL	`End/Ctrl-E`
Move characterwise	`Left/Ctrl-B or Right/Ctrl-F`
Move wordwise	`Alt-Left or Alt-Right`
Move through directory listing	`on empty CMD line: Alt-Left or Alt-Right`
Search history for prefix in CMD line	`Up or Down`
Search history for token containing token under cursor	`Alt-Up or Alt-Down`
Delete characterwise	`Delete/Ctrl-D (forwards) or Backspace (backwards)`
Delete entire line	`Ctrl-C`
Move contents from cursor to EOL to killring	`Ctrl-K`
Move contents from BOL to cursor to killring	`Ctrl-U`
Repaint Screen	`Ctrl-L`
Move previous word to killring	`Ctrl-W`
Move next work to killring	`Alt-D`
Print description of CMD under cursor	`Alt-W`
List contents of current directory or directory under cursor	`Alt-L`
Add '|less;' to end of job under cursor	`Alt-P`
Capitalize current word	`Alt-C`
Make current word uppercase	`Alt-U`