

Nicely Formatted Latex Version

You can find a better cheat sheet here:
https://github.com/michael-younkin/fish_shell_cheatsheet

Fish Syntax

Variables

Three kinds: universal, global, and local. Universal variables are shared btw. all sessions on the computer per user. Global variables are specific to the current fish session, but they are outside of any block scope. Local variables are specific to a particular block scope and are automatically erased.

Set a variable as universal with `-U`, as global with `-g`, or local with `-l`. Scoping rules are as follows:

1. If a variable is explicitly set to either universal, global or local, that setting will be honored. If a variable of the same name exists in a different scope, that variable will not be changed.
2. If a variable is not explicitly set to be either universal, global or local, but has been previously defined, the variable scope is not changed.
3. If a variable is not explicitly set to be either universal, global or local and has never before been defined, the variable will be local to the currently executing function. Note that this is different from using the `-l` or `-local` flag. If one of those flags is used, the variable will be local to the most inner currently executing block, while without these the variable will be local to the function. If no function is executing, the variable will be global.

Exporting Variables

Export a variable with `-x`.

Arrays

Store multiple strings in one variable with an array. Access an index:

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



By myounkin
cheatography.com/myounkin/

Fish Syntax (cont)

Iterate:

```
for i in $PATH; echo $i
```

Definition:

```
set smurf blue small
makes an array called smurf
containing "blue" and "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

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

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`.

put the suspended job in the background with `bg`. You can

list all running jobs with `jobs`.

Chaining Commands

Each command ends in either a newline or a semicolon.

Chain commands using `command; command2` or `command; or command2`.

and `and` or `or` check the command's exit status and act accordingly.

Aliases

To define an alias, either make a function:

```
function ls
  command ls --color=auto $argv
end
```

...or use `use alias NAME DEFINITION` which does the same for you.

IO Redirection and Piping

Redirect

```
>> $stdin
```

```
>> $stdout
```

```
>> $stderr
```

```
>> $dest
```

Expansion (cont)

A comma separated list of characters enclosed in curly braces is expanded to each element of the list is a new parameter:

```
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 the variable with braces to separate it from text.

Process Expansion

% followed by a string is expanded into a PID according to the following rules:

1. If the string is self, insert the shell's PID
2. If the string is the ID of a job, insert the process group ID of 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. If none of the above matches, then 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
```

Variables

argv	array of arguments to a shell function; only defined in a function call or when fish is invoked to run a script
history	array containing the last commands that were entered
HOME	the user's home directory
PWD	the current working directory
status	the exit status of the last foreground job to exit

Variables (cont)

USER	the current username
------	----------------------

Command line editor

Complete current token

```
Tab
```

Accept autosuggestion

```
at end of line: End/Ctr rl- E/R igh t/Ctr l-F
```

Move to beginning of line

```
Home/C trl-A
```

Move to end of line

```
End/Ctrl-E
```

Move character-wise

```
Left/C trl-B or Right/ Ctrl-F
```

Move word-wise

```
Alt-Left or Alt-Right
```

Move through directory history

```
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 beginning of line to cursor to killring

```
Ctrl-U
```

Repaint screen

```
Ctrl-L
```

Move previous word to killring

```
Ctrl-W
```

Move next word to killring

```
Alt-D
```

Command line editor (cont)

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

```
t/Ctr l-F
```

Capitalize current word

```
Alt-C
```

Make current word uppercase

```
Alt-U
```

These shortcuts can be changed using the bind built-in command.

