Contents

Setup **2**

Fish-specific keyboard shortcuts **3**

If statement **4**

Switch statement **5**

Functions **6**
argv 6
Only define undefined functions 6

and, or **7**
A safer way to cat 7
A safer mv 7

prompt **8**
Verbose prompt 8
Simplified prompt 8

Miscellaneous **10**
Single quotes and double quotes 10

Setup

set fish as the default system shell in fish

```
chsh -s (which fish)
```

configuration entry point

```
vim ~/.config/fish/config.fish
```

sourcing

```
source ~/git/dot-files/fish/3fishrc-root-macbook
```

disable svn for good

```
function svn
end
```

Fish-specific keyboard shortcuts

`<c-f>` same as right arrow key, to accept auto-suggestions or move right

`<c-b>` same as left arrow key

`<c-p>` same as upper arrow key

`<c-n>` same as lower arrow key

`<c-l>` or `<command-r> on mac` clean screen better than **clear**

`<c-i>` same as `<tab>`

`bind` see all fish command line shortcuts

If statement

```
set -xU I_AM_TRUE 1
if test -n "$I_AM_TRUE"
  echo "now you see me"
end
if test -n "$I_AM_FALSE"
  echo "now you don't"
end
```

Switch statement

```
function showmethedir
  switch $PWD
    case $GITHUB_DIR/learn/learn-linux/command-line/fish
      echo 'this is fish/ directory haha~'
    case $GITHUB_DIR/learn/learn-linux/command-line
      echo 'this is command-line/ directory oh yeah~'
    case '*'
      echo 'I do not recognize this directory'
  end
end
cd $GITHUB_DIR/learn/learn-linux/
showmethedir #I do not recognize this directory
cd $GITHUB_DIR/learn/learn-linux/command-line
showmethedir #this is command-line/ directory oh yeah~
cd $GITHUB_DIR/learn/learn-linux/command-line/fish
showmethedir #this is fish/ directory haha~
```

Functions

argv

`$argv[1]` is the **first** element, not the second.

```
function func
  echo $argv[1]+$argv[3]
end
func 1 2 3 # 1+3
```

`$argv` can also be used as a whole.

```
function vimvanilla
  vim -u NONE -N $argv
end
vimvanilla $GITHUB_DIR/learn/learn-linux/command-line/fish/fish.md +55 #this line
```

Only define undefined functions

```
# new definition wins
function ilikenewdefinition
  echo 'hello world'
end
function ilikenewdefinition
  echo 'another definition'
end
ilikenewdefinition
# old definition wins
type onlydefinemeonce > /dev/null 2>&1; or function onlydefinemeonce
  echo 'hello world'
end
type onlydefinemeonce > /dev/null 2>&1; or function onlydefinemeonce
  echo 'you no see me'
end
onlydefinemeonce
```

and, or

`and` only runs when the previous command returns true.

`or` only runs when the previous command returns false.

A safer way to cat

```
# nasty error message if file not exist
cat resources/file_that_may_not.exist
# no error message
test -s resources/file_that_may_not.exist; and cat resources/file_that_may_not.exist
```

A safer mv

```
function asafermvsetup
  echo 'important content' > resources/asafermv
  echo 'new content' > resources/asafermv2
end
asafermvsetup
#
# the previous content in asafermv is then lost
mv resources/asafermv2 resources/asafermv
cat resources/asafermv # new content
#
function safermv
  test -s $argv[2]; or mv $argv[1] $argv[2]
end
# the previous content in asafermv is preserved
asafermvsetup
safermv resources/asafermv2 resources/asafermv
cat resources/asafermv # important content
```

But of course, the best way is always mv -i which prompts on conflict

```
mv -i resources/asafermv2 resources/asafermv
```

prompt

Verbose prompt

a prompt that looks like: `tangke@Kirks-MacBook-Pro ~/g/active-notes:master`, or `tangke@Kirks-MacBook-Pro ~/g/l/l/command-line:master>`

```
# red for dirty git branches
set fish_git_dirty_color red
# green for clean git branches
set fish_git_not_dirty_color green
# process git status
type parse_git_branch > /dev/null 2>&1; or function parse_git_branch
  # define branch name
  set -l branch (git branch 2> /dev/null | grep -e '\* ' | sed 's/^..\(.*\)/\1/')

  # define git_diff (we only care about whether it's empty)
  set -l git_diff (git diff)

  # apply different colors based on whether the git branch is dirty
  if test -n "$git_diff"
    echo (set_color $fish_git_dirty_color)$branch(set_color normal)
  else
    echo (set_color $fish_git_not_dirty_color)$branch(set_color normal)
  end
end
function fish_prompt
  set -l git_dir (git rev-parse --git-dir 2> /dev/null)
  if test -n "$git_dir"
    printf '%s@%s %s%s%s:%s> ' (whoami) (hostname|cut -d . -f 1) \
      (set_color $fish_color_cwd) (prompt_pwd) (set_color normal) (parse_git_branch)
  else
    printf '%s@%s %s%s%s> ' (whoami) (hostname|cut -d . -f 1) \
      (set_color $fish_color_cwd) (prompt_pwd) (set_color normal)
  end
end
```

Simplified prompt

a slightly simpler prompt that looks like: `~/g/active-notes:master`, or

```
~/g/l/learn-linux:master>
```

```
# red for dirty git branches
```

```
set fish_git_dirty_color red
# green for clean git branches
set fish_git_not_dirty_color green
# process git status
type parse_git_branch > /dev/null 2>&1; or function parse_git_branch
  # define branch name
  set -l branch (git branch 2> /dev/null | grep -e '\* ' | sed 's/^..\(.*\)/\1/')

  # define git_diff (we only care about whether it's empty)
  set -l git_diff (git diff)

  # apply different colors based on whether the git branch is dirty
  if test -n "$git_diff"
    echo (set_color $fish_git_dirty_color)$branch(set_color normal)
  else
    echo (set_color $fish_git_not_dirty_color)$branch(set_color normal)
  end
end
function fish_prompt
  set -l git_dir (git rev-parse --git-dir 2> /dev/null)
  if test -n "$git_dir"
    printf '%s%s%s:%s> ' (set_color $fish_color_cwd) (prompt_pwd) \
      (set_color normal) (parse_git_branch)
  else
    printf '%s%s%s> ' (set_color $fish_color_cwd) (prompt_pwd) (set_color normal)
  end
end
```

Miscellaneous

Single quotes and double quotes

use `'` or `"` to escape spaces for file names and grep key words. In bash, `'` and `"` are slightly different, as `"` doesn't prevent special characters from expanding, but in fish they are identical (which is really nice).