Ruby Fundamentals

Project

Create a cheat sheet that you can run in the command line to help you remember key terms and shortcuts

Demo!

What we will cover

- Data types
- Variable declaration
- Scope
- String interpolation
- Methods
- Conditionals
- Loops
- Executing shell commands

What we will cover (cont.)

- Man pages
- irb
- Array index
- Method chaining
- Ruby docs

Data types

- Integer
- Float
- Boolean
- String
- Arrays, Hashes, Objects …

Everything in Ruby is an object. Demo!

Variables

Declaring variables in ruby:

```
x = 1
x = ‘hello world’
x = true
x = 1.4
x = 1 + 25
x = ‘hello’ + ‘ world’
```

Types of variables

Foo = ‘I am a constant’

- Capital 1st letter
- Scoped to the file
- Can be changed but should not be (ruby will warn you)

foo = ‘I am a locally scoped variable’

- Lowercase 1st letter & underscore case my_var
- Scoped to where it is defined for example in a method

@foo = ‘I am an instance variable

- Starts with an @ symbol
- Scoped to the class or running instance

Scope

Foo = 1, foo = 2, @foo = 3

What will be printed in each of the following examples

```
puts Foo
puts foo
puts @foo
puts Foo + foo
foo = Foo + foo
puts foo + @foo
```

```
def print_foo
  puts Foo
  puts @foo
  puts foo
end

print_foo
```

```
def print_foo(bar)
  puts Foo
  puts @foo
  puts bar
end

print_foo(2)
```

Foo = 1, foo = 2, @foo = 3

```
puts Foo
>> 1
puts foo
>> 2
puts @foo
>> 3
puts Foo + foo
>> 3
foo = Foo + foo
puts foo + @foo
>> 6
```

```
def print_foo
  puts Foo
  puts @foo
  puts foo
end

print_foo
>> 1
>> 3
>> `undefined…`
```

```
def print_foo(bar)
  puts Foo
  puts @foo
  puts bar
end

print_foo(foo)
>> 1
>> 3
>> 2
```

String Interpolation

2 ways to declare strings

puts ‘hello world’

puts “hello world”

‘so what\’s the difference’

“so what’s the difference”

Interpolation

```
greeting = ‘hello’
name = ‘world’
puts “hello #{name}”
puts “#{greeting} #{name}”
```

DEMO!

Methods

- breaks our code into manageable chunks
- should perform a single task
- should have a concise name
- should perform an action or return an object
- always returns last line executed
- if you have to use and / or to describe what your method does you may have 2 methods

Methods (cont)

```
def hello_world
  puts ‘hello world’
end

def hello(planet)
  puts “hello #{planet}”
end

def tripple_my_number(number)
  number * 3
end
```

```
def odd_or_even(number)
  if number % 2 == 0
    ‘even’
  else
    ‘odd’
  end
end

puts odd_or_even(tripple_num(3))
```

Conditionals

- if, elsif, else
- case
- ternary

if, elsif, else

```
if x == true
  puts x
end

if x
  puts x
end

if x
 puts “It’s true”
else
 puts “It’s a lie”
end
```

```
if num <= 0
  puts “The number is too low”
elsif num > 3 && num < 7
  puts “The number is just right”
else
  puts “The number is too high”
end
```

```
if num % 2 == 0
  if num < 10
    puts “even less than 10”
  else
    puts “even greater than 10”
  end
else
  puts “The number is odd”
end
```

case (review)

```
case number
  when 1,2
    puts number
  when 3
    puts ‘THREE’
  else
    puts ‘Too high’
end
```

Ternary

Before

```
if num % 2 == 0
  a = true
else
  a = false
end
```

After

```
a = num % 2 == 0 ? true : false

a = num % 2 == 0 ? (num + 1) : ( num + 2)

assignment = condition ? if : else
```

Connecting a few ...

```
number = 2

def alter(num)
  num % 2 == 0 ? (num + 1) : (num + 3)
end

new_number = alter(number)
new_new_number = alter(alter(new_number))
puts alter(new_new_number)
```

What will be the outcome?

Loops and modifiers

- while
- until
- for
- each
- next
- break
- retry

While

```
x = 0
while x <= 3
  puts x
  x += 1
end

puts x + 1
```

```
x = 0  >> 0
x = 1  >> 1
x = 2  >> 2
x = 3  >> 3
x = 4  >> 5
```

```
while condition is true
  code
  modify condition
end
```

What happens if condition is not modified in loop?

Until

```
x = 0
until x > 3
  puts x
  x += 1
end


puts x + 1
```

```
x = 0  >> 0
x = 1  >> 1
x = 2  >> 2
x = 3  >> 4
```

```
until condition is true
  code
  modify condition
end
```

For

```
for x in (0..5)
  puts x
end
```

```
x = 0  >> 0
x = 1  >> 1
x = 2  >> 2
x = 3  >> 3
x = 4  >> 4
x = 5 >> 5
```

```
for variable in condition
  code
end
```

Each

```
(0..5).each do |x|
  puts x
end
```

```
(0..5).each { |x| puts x }
```

```
x = 0  >> 0
x = 1  >> 1
x = 2  >> 2
x = 3  >> 3
x = 4  >> 4
x = 5 >> 5
```

```
for variable in condition
  code
end
```

Next

```
x = 0
(0..5).each do |x|
  if x % 2 == 0
     y = ‘even’
  else
   y = ‘odd’
   next
  end
   puts x
end
```

What will print for odd numbers?

What about even?

Break

```
x = 0
while x <= 10
  break if x == 5
  puts x
  x += 1
end
```

Retry

```
x = 0
while x <= 10
  retry if x == 2
  puts x
  x += 1
end
```

Oops! Infinite loop

Nested loops

```
x = 0
y = 0

(1..5).each do |i|
  puts ‘in x loop’
  x += i
  (1..2).each do |j|
    y += j
  end
end
```

How many times does outer loop iterate?

How many times does inner loop iterate?

Shell commands

Whenever you enter a command in your terminal an action is performed.

```
ls Users/dave
```

will print a list of files in the directory Users/dave

Shell commands (cont)

To run a shell command while in a ruby script wrap it in backticks ` `

```
`ls /Users/dave`
```

executes the shell command

```
puts `ls /Users/dave`
```

prints the list of files in the directory

```
files = `ls /Users/dave`
```

saves the list of files in the directory to a variable called file

```
array = `ls /Users/dave`.split(“\n”)
```

saves each file as a cell in an array [‘file1’, ‘file2’, …]

Man pages

Your terminal is equipped with manual pages for all native commands.

If I want more information on moving a file I can type *man mv* and I will see all the docs on the mv command including example usage.

Pro tip for today's project:

```
puts `man #{cmd}`
```

irb

Demo!

Array index (refresher)

```
arr = [‘A’, ‘B’, ‘C’, ‘D’]
puts arr[1]
puts arr[0]
puts arr[3]
puts arr[4]
puts arr.first
puts arr.last
```

Method chaining

```
str = “Hello class don’t fall asleep”
puts str.split(“ “).last
puts str.downcase.split(“ “).join(‘-’)
```

Don’t get carried away!

Creating / running a ruby script

A ruby script should be contained in a file with a .rb extension

e.g. my_script.rb

To run the script from console type ruby <file>

e.g. ruby my_script.rb

Ruby docs

http://ruby-doc.org

Bookmark NOW!!!

Questions?

Project:

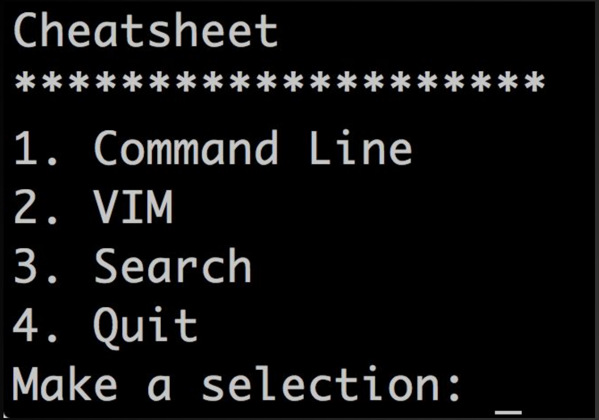
Create a cheat sheet project to help learn terminal commands.

Objectives:

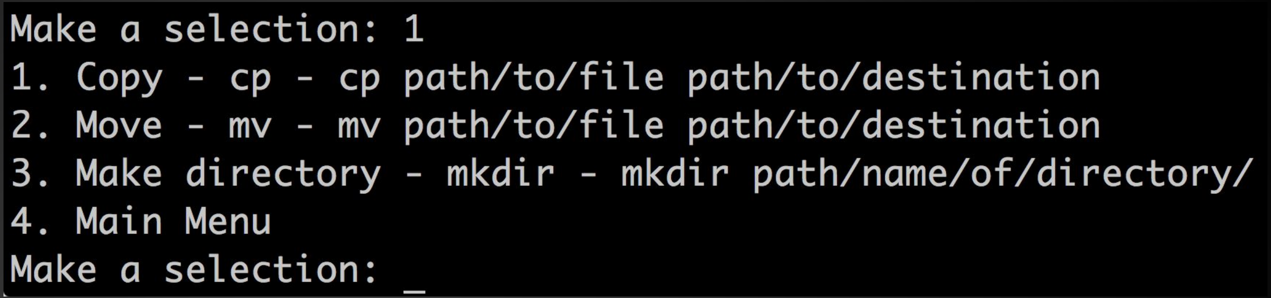
1. Create a menu that takes in a user input
   a. 1. Command Line 2. IDE 3. Search 4. Quit
   b. Based on user choice go to a 2nd menu
2. When command line is chosen display a list of command line options
   a. Also supply a way for the user to get back to the main menu
   b. When a command line menu option is selected display the man pages for that option
3. When the IDE menu is chosen list shortcut options
   a. When a shortcut is chosen display more information about the shortcut
   b. Also provide a way for the user to get back to the main menu

BONUS:

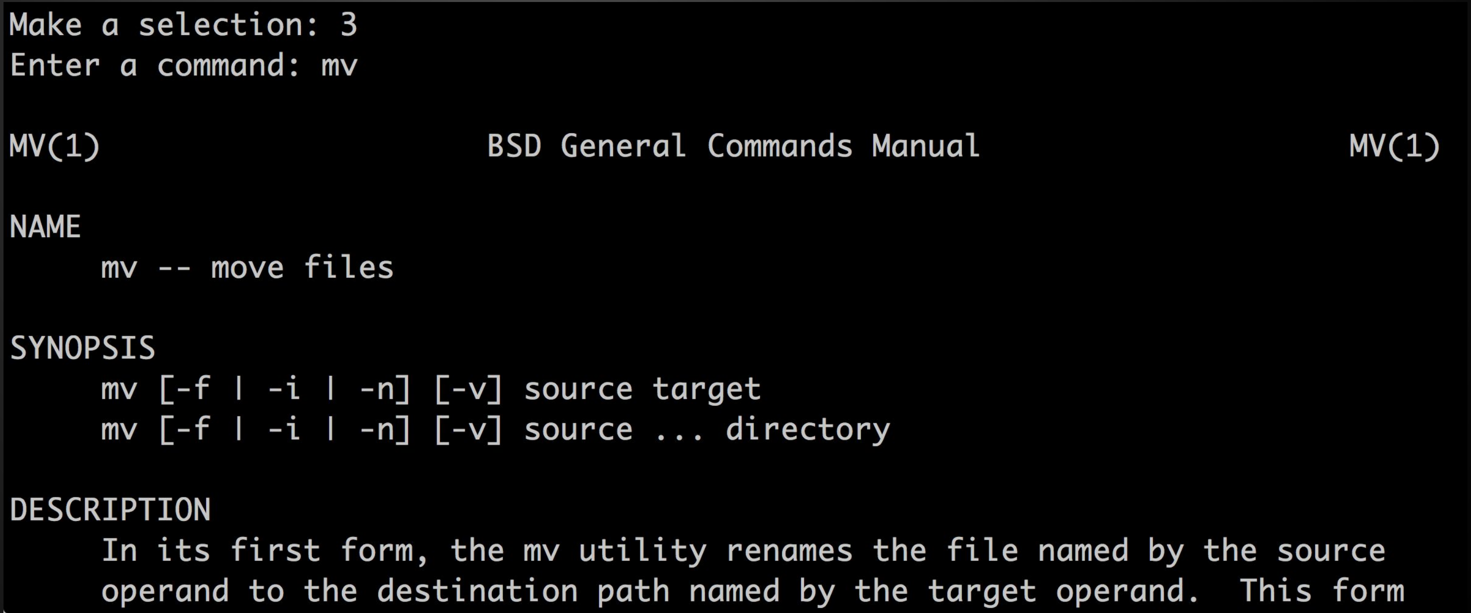
Allow the user to search. When a command is entered into the search show the man pages for that command.



```
Cheatsheet
*********************
1. Command Line
2. VIM
3. Search
4. Quit
Make a selection: _
```



```
Make a selection: 1
1. Copy - cp - cp path/to/file path/to/destination
2. Move - mv - mv path/to/file path/to/destination
3. Make directory - mkdir - mkdir path/name/of/directory/
4. Main Menu
Make a selection: _
```



```
Make a selection: 3
Enter a command: mv

MV(1)                     BSD General Commands Manual                    MV(1)

NAME
     mv -- move files

SYNOPSIS
     mv [-f | -i | -n] [-v] source target
     mv [-f | -i | -n] [-v] source ... directory

DESCRIPTION
     In its first form, the mv utility renames the file named by the source
     operand to the destination path named by the target operand.  This form
```