
Web Application Testing in Ruby
With Selenium and friends.

Web Application Testing in Ruby

With Selenium and friends.

Željko Filipin

This book is for sale at http://leanpub.com/watirbook

This version was published on 2014-03-08

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Željko Filipin by spreading the word about this book on Twitter!

The suggested hashtag for this book is #watirbook.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#watirbook

Contents

About the Book

Web Application Testing in Ruby (with Selenium and friends)[1] is a book on, well, web application testing in Ruby[2] (Watir). It is not finished. We are working on it.

Book source code is at GitHub[3]. You can discuss the book at Google Groups[4]. More information about the book is available at filipin.eu/tag/watirbook[5].

The book is available as PDF[6], EPUB[7] (iPad, iPhone, iPod) and MOBI[8] (Kindle) file at leanpub.com/watirbook[9]. HTML version of the book is at leanpub.com/watirbook/read[10]. Suggested price is $9.99, but you can get the book for free! Click **Buy the ebook now!** button and move the **You pay** slider to the left hand side until the amount goes down to **$0.00**. Of course, you can move the slider to the right hand side and pay more for the book. All money goes to the Watir project.

Money that book readers have donated to the Watir project == $1,523.97 (Updated 2014-01-25. For more stats see stats page[11].)

---

[1] https://leanpub.com/watirbook
[2] http://watir.com/
[3] https://github.com/watir/watirbook
[4] http://groups.google.com/group/watirbook/
[5] http://filipin.eu/tag/watirbook/
[6] http://en.wikipedia.org/wiki/Portable_Document_Format
[7] http://en.wikipedia.org/wiki/EPUB
[8] http://en.wikipedia.org/wiki/Mobipocket
[9] https://leanpub.com/watirbook
[10] https://leanpub.com/watirbook/read
[11] https://github.com/watir/watirbook/blob/master/misc/stats.md

Installation

Installation is not complicated, but unfortunately, it is not trivial either.

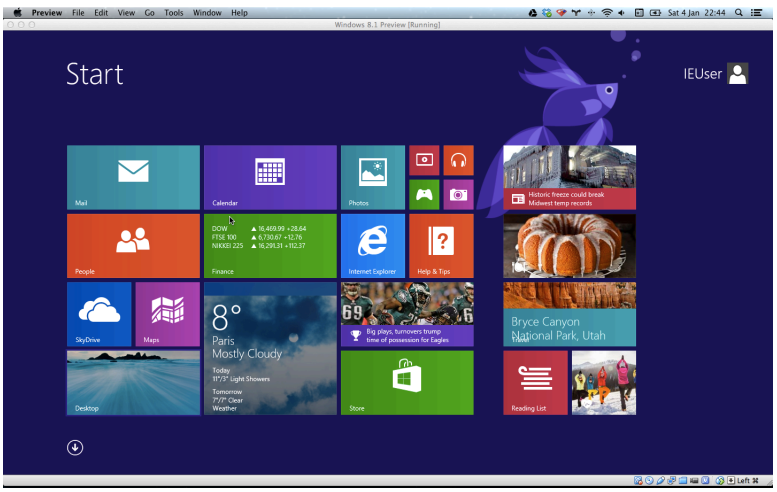
The tools that we will be using in the book are available on Windows, Mac and Linux, and each of them has several releases currently in use (or in case of Linux, both distributions and releases). Just the most recent releases of each operating system will be covered in the book.

Everything is tested using VirtualBox 4.3.6 virtual machines, except Mac OS, since it does not want to be virtualized. Host OS for VirtualBox is Mac OS X 10.9.1.

You will probably be bored to tears if you read all installation chapters. A lot of stuff is repeated. Read only the chapters you need.

Windows 8.1

You will need internet access if you want to follow examples. All examples are tested with Microsoft Windows 8.1 Pro. All browsers are English (US) version.



Windows 8.1 default desktop

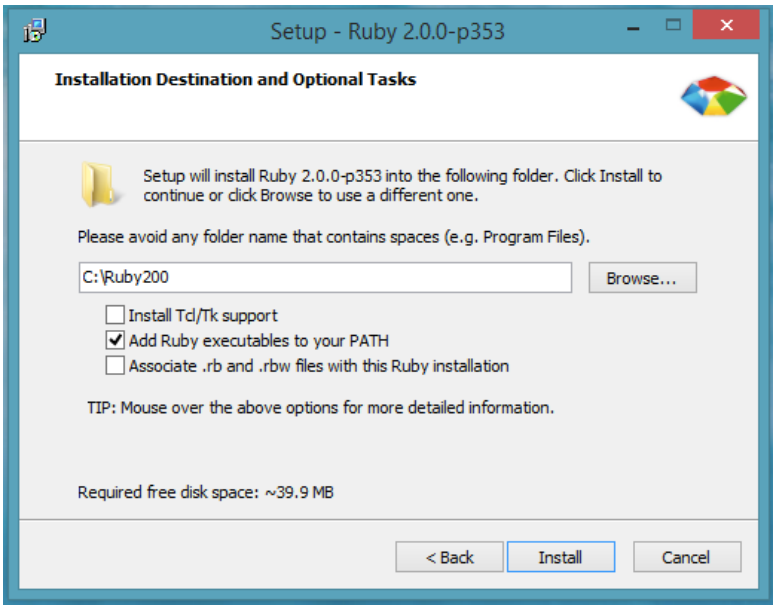
Ruby

You probably do not have Ruby installed. To make sure, open Command Prompt application and type `ruby -v`. If you are not familiar with Command Prompt, see *Command-line interface* chapter.

```
>ruby -v
'ruby' is not recognized as an internal or
external command, operable program or batch file.
```

If you get the same thing as I did, you do not have Ruby installed.

Download the latest Ruby 2.0 from *rubyinstaller.org/downloads*[12]. At the moment it is *Ruby 2.0.0-p353*. Execute the file.



Ruby Installation

You can leave all settings at default values, except at the *Installation Destination and Optional Tasks* screen check *Add Ruby executables to your PATH* checkbox. Installation should take you just a few seconds.

Let's check if Ruby is installed. You will have to open another command prompt, because the one you have opened does not see Ruby.

```
>ruby -v
ruby 2.0.0p353 (2013-11-22) [i386-mingw32]
```

Congratulations! You now have the latest and greatest Ruby!

RubyGems

Software written in Ruby is usually distributed as RubyGems (colloquial name is *gem*), Ruby package manager. Sometimes Ruby installations do not have the latest versions of RubyGems, so we will first update it. RubyGems is also a gem, (a bit recursive, right?) and we get it's version with `gem -v`.

[12] http://rubyinstaller.org/downloads

```
>gem -v
2.0.14
```

You should update it with `gem update --system`:

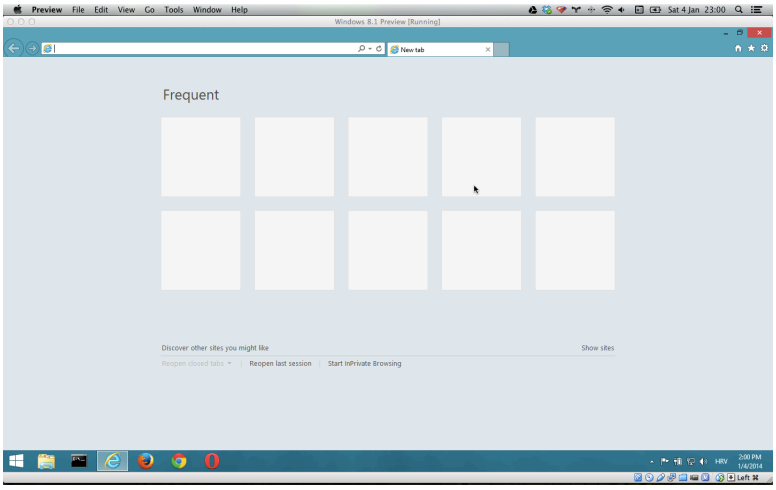
```
>gem update --system
(...)
RubyGems system software updated
```

selenium-webdriver

Let's try selenium-webdriver gem.

```
>gem install selenium-webdriver --no-ri --no-rdoc
(...)
Successfully installed selenium-webdriver-2.38.0
(...)
```

Internet Explorer

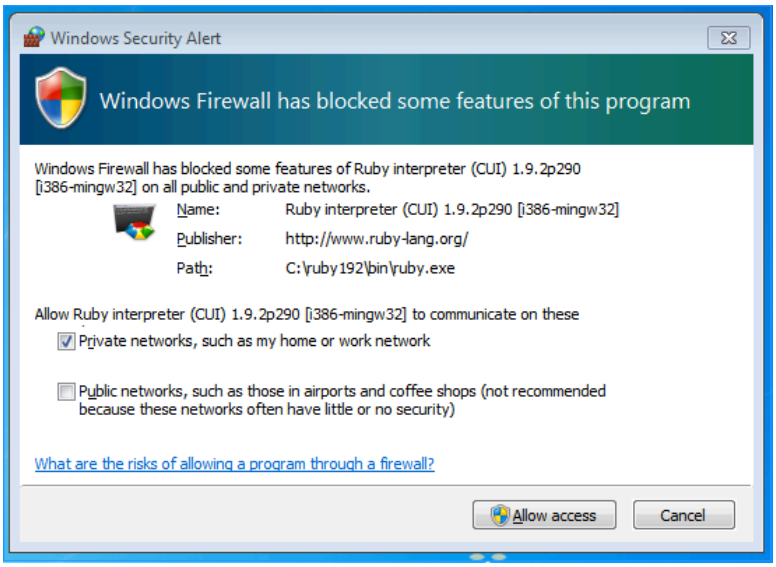


Internet Explorer 11

Since Internet Explorer (tested with version 11) is already installed, we will start with it. We will be using IRB (Interactive Ruby Shell). If you are not familiar with it, see *IRB* chapter.

```
>irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for
:internet_explorer
Selenium::WebDriver::Error::WebDriverError: Unable
to find standalone executable. Please download the
IEDriverServer from
http://code.google.com/p/selenium/downloads/list
and place the executable on your PATH.
(...)
```

Windows Firewall popup appeared letting me know that it has blocked Ruby.



Windows Firewall has blocked some features of this program

For now just close the popup, let's see how to fix the error message. Follow the instructions from the error message. Go to code.google.com/p/selenium/downloads/list[13] and download 32-bit or 64-bit IEDriverServer. Extract the downloaded zip file (with mouse right click and then *Extract All..*, for example). Let's find out what is on the PATH.

```
>PATH
PATH=C:\Windows\system32;C:\Windows;
C:\Windows\System32\Wbem;
C:\Windows\System32\WindowsPowerShell\v1.0\;
C:\Ruby200\bin
```

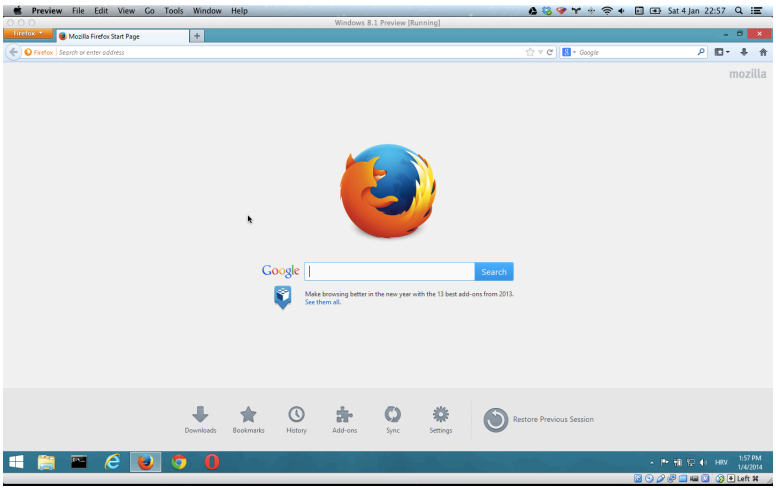
`C:\Ruby200\bin` (or where ever you have installed Ruby) looks like a good place to me. Move the IEDriverServer file there.

[13] http://code.google.com/p/selenium/downloads/list

Let' try again:

```
>irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for
:internet_explorer
Started InternetExplorerDriver server (32-bit)
2.38.0.0
Listening on port 5555
=> #<Selenium::WebDriver::Driver:0x5469141e
browser=:internet_explorer>

> browser.get "http://watir.com"
=> nil
```

Firefox



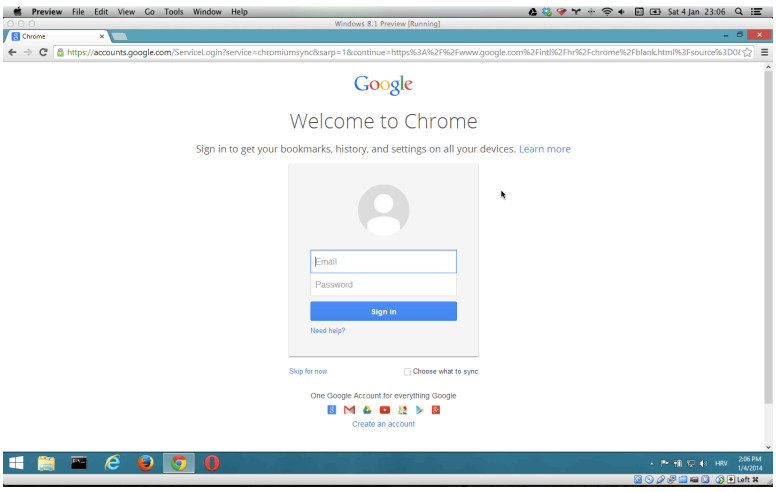
Firefox 26

If it is not installed (tested with version 25.0.1), download it from mozilla.com/firefox[14].

[14] http://www.mozilla.com/firefox/

```
>irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :firefox
=> #<Selenium::WebDriver::Driver:0xdc8ad4a
browser=:firefox>

> browser.get "http://watir.com"
=> ""
```

Chrome



Chrome 31

If it is not installed (tested with version 31), download it from google.com/chrome[15].

```
>irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :chrome
Selenium::WebDriver::Error::WebDriverError: Unable
to find the chromedriver executable. Please
download the server from
http://chromedriver.storage.googleapis.com/
index.html and place it somewhere on your PATH.
More info at
http://code.google.com/p/selenium/wiki/
```

[15] http://www.google.com/chrome

```
ChromeDriver.
(...)
```

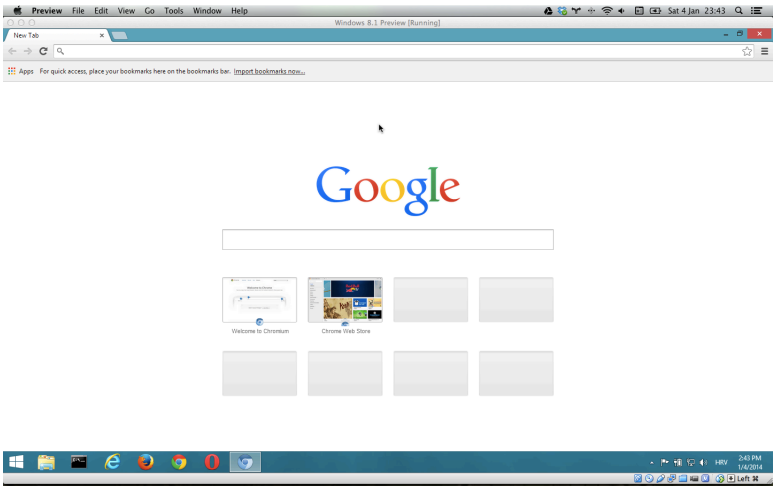
You have to download ChromeDriver[16]. Unzip the file and put it in any folder that is in your PATH. If you do not know what is PATH, see *Internet Explorer* chapter. In short, move the file to Ruby folder, probably C:\Ruby200\bin.

Let's try again:

```
>irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :chrome
Starting ChromeDriver (v2.7.236900) on port 9515
=> #<Selenium::WebDriver::Driver:0x..fc23f2ebe
browser=:chrome>

> browser.get "http://watir.com"
=> nil
```

I got Windows Firewall popup again letting me know that it has blocked ChromeDriver. Just close it for now, I have no idea what to do with it.

Chromium



Chromium 34

First make sure that you can drive Chrome. (See *Chrome* chapter.) Then download Chromium[17]. (Tested with version 33.) Replace C:\chrome\chrome.exe with path to Chromium executable.

[16] http://chromedriver.storage.googleapis.com/index.html
[17] https://download-chromium.appspot.com/

```
>irb

> require "selenium-webdriver"
=> true

> Selenium::WebDriver::Chrome.path =
'C:\chrome\chrome.exe'
=> "C:\\chrome\\chrome.exe"

> browser = Selenium::WebDriver.for :chrome
Starting ChromeDriver (v2.7.236900) on port 9515
=> #<Selenium::WebDriver::Driver:0x..f9d38b82e
browser=:chrome>

> browser.get "http://watir.com"
=> nil
```

PhantomJS

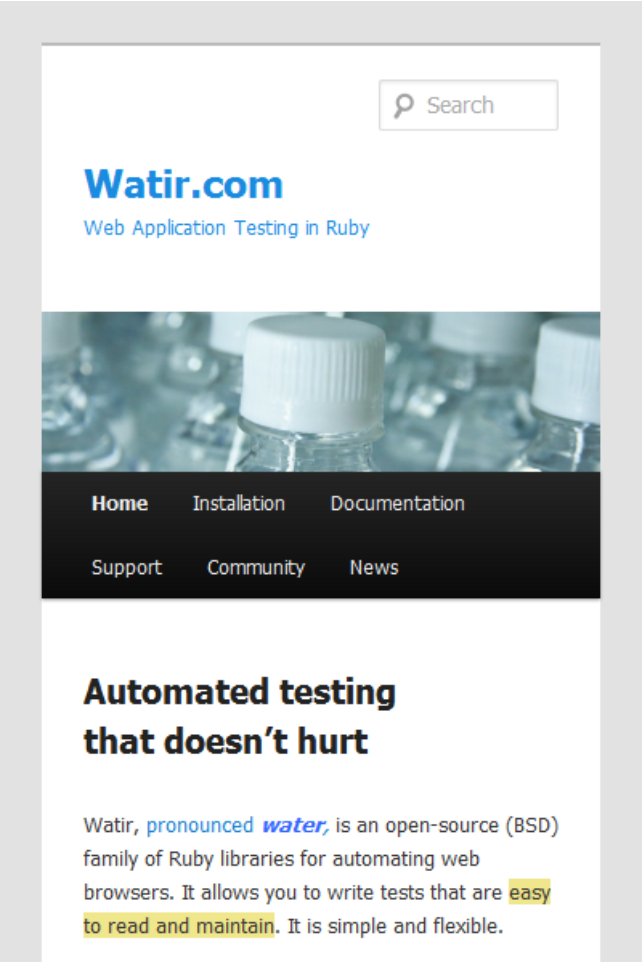
To drive PhantomJS[18] (tested with version 1.9.2) download it, unzip the file and put it in any folder that is in your `PATH`. If you do not know what is `PATH`, see *Internet Explorer* chapter. In short, move `phantomjs.exe` file to Ruby folder, probably `C:\Ruby200\bin`.

Let's try driving it:

```
>irb

> browser = Selenium::WebDriver.for :phantomjs
(...)
=> #<Selenium::WebDriver::Driver:0x442ed372
browser=:phantomjs>

> browser.get "http://watir.com"
=> {}

> browser.save_screenshot "phantomjs.png"
=> #<File:phantomjs.png (closed)>
```

The last command saves screenshot of the page. A screenshot from a headless browser. Nice, right?

[18] http://phantomjs.org/



PhantomJS

Java

To drive Opera, you will have to install Java first. Let's check if Java is already installed with `java -version`:

```
1 >java -version
2 'java' is not recognized as an internal or
3 external command, operable program or batch file.
```

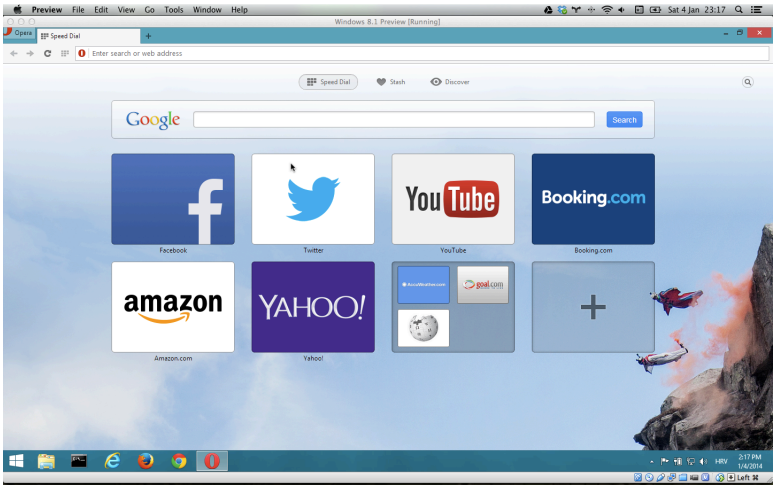
Looks like we will have to install Java. There is big *Free Java Download* button at java.com[19]. Execute downloaded file and install Java. Let's check if Java is really installed with `java -version`:

[19] http://www.java.com/

```
>java -version
java version "1.7.0_45"
Java(TM) SE Runtime Environment
(build 1.7.0_45-b18)
Java HotSpot(TM) Client VM (build 24.45-b08,
mixed mode, sharing)
```

Looks good to me!

Opera



Opera 18

And finally, let's drive Opera. If you do not have it installed, you can get it at opera.com[20] (tested with version 18.0).

```
>irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
Selenium::WebDriver::Error::WebDriverError: Unable
to find the Selenium server jar. Please download
the standalone server from
http://code.google.com/p/selenium/downloads/list
and set the SELENIUM_SERVER_JAR environmental
variable to its location. More info at
http://code.google.com/p/selenium/wiki/OperaDriver.
(...)
```

[20] http://www.opera.com/

Download `selenium-server-standalone-2.39.0.jar` (or newer version) from Google Code[21] and put it in `C:\Ruby200\bin`. Then make `SELENIUM_SERVER_JAR` environmental variable and set it to `C:\Ruby200\bin\selenium-server-standalone-2.39.0.jar`. (If you have a newer version of `selenium-server-standalone` file, replace `2.39.0` appropriately.)

If you just want to try driving Opera, typing this into Command Prompt will do the trick:

```
>set SELENIUM_SERVER_JAR=C:\Ruby200\bin\
selenium-server-standalone-2.39.0.jar
```

Let's try driving Opera again.

```
>irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
Selenium::WebDriver::Error::UnknownError: Could
not find a platform that supports bundled
launchers, please set it manually
Build info: version: '2.39.0', revision: 'ff23eac',
time: '2013-12-16 16:11:15'
System info: host: 'IE11WIN8_1', ip: '10.0.2.15',
os.name: 'Windows 8',
os.arch: 'x86', os.version: '6.2', java.version:
'1.7.0_45'
Driver info: driver.version: OperaDriver
(org.openqa.selenium.WebDriverException)
```

Looks like Selenium can not drive Opera on Windows 8.1.

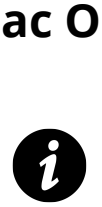
If I ever manage to drive Opera on Windows, to create a permanent environmental variable, use `setx`:

```
>setx SELENIUM_SERVER_JAR C:\Ruby200\bin\
selenium-server-standalone-2.39.0.jar
SUCCESS: Specified value was saved.
```

Open new command prompt, the old one will not see `SELENIUM_SERVER_JAR` variable.

[21] http://code.google.com/p/selenium/downloads/list

Mac OS X 10.9



You will need internet access if you want to follow examples. All examples are tested with Mac OS X 10.9. All browsers are English (US) version.

Mac OS X 10.9 default desktop

Ruby

Good news is that Ruby is already installed by default. To check Ruby version, open Terminal application any type `ruby -v`. If you are not familiar with Terminal, see *Command-line interface* chapter.

You should get this:

```
$ ruby -v
ruby 2.0.0p247 (2013-06-27 revision 41674)
[universal.x86_64-darwin13]
```

RubyGems

It is already installed, but an old version. Let's see which version is here with `gem -v`.

You should get this:

```
$ gem -v
2.0.3
```

Fortunately, it is easy to upgrade RubyGems with `sudo gem update --system`:

```
$ sudo gem update --system
(...)
RubyGems 2.1.11 installed
(...)
```

selenium-webdriver

Let's try selenium-webdriver gem. Install it with `sudo gem install selenium-webdriver --no-ri --no-rdoc`.

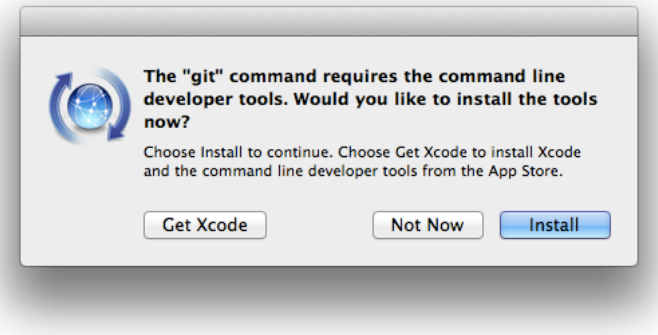
You will probably get this:

```
$ sudo gem install selenium-webdriver
(...)
Fetching: ffi-1.9.3.gem (100%)
Building native extensions.  This could take a
while...
ERROR:  Error installing selenium-webdriver:
ERROR: Failed to build gem native extension.
(...)
```

Fortunately, it is easy to fix. Install command line developer tools. To install them, type this into Terminal.

```
$ git --version
```

A popup will appear asking if you would like to install command line developer tools.

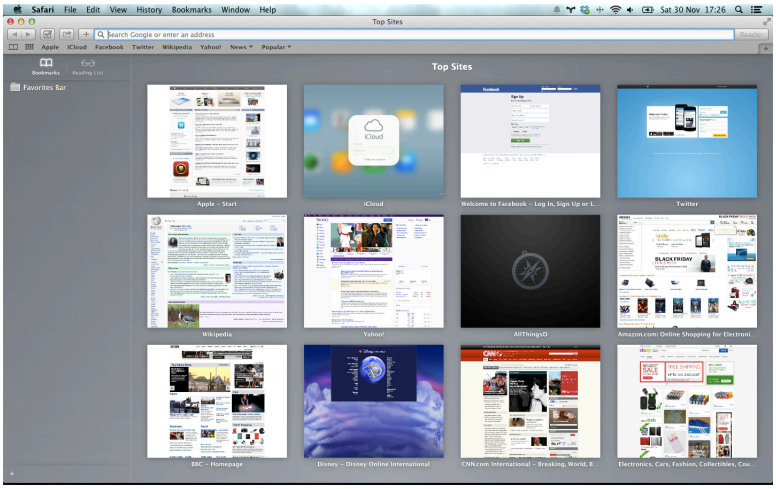


Install command line developer tools popup

Click *Install*. After the installation is finished, try again:

```
$ sudo gem install selenium-webdriver --no-ri
--no-rdoc
(...)
Successfully installed selenium-webdriver-2.37.0
(...)
```

Safari

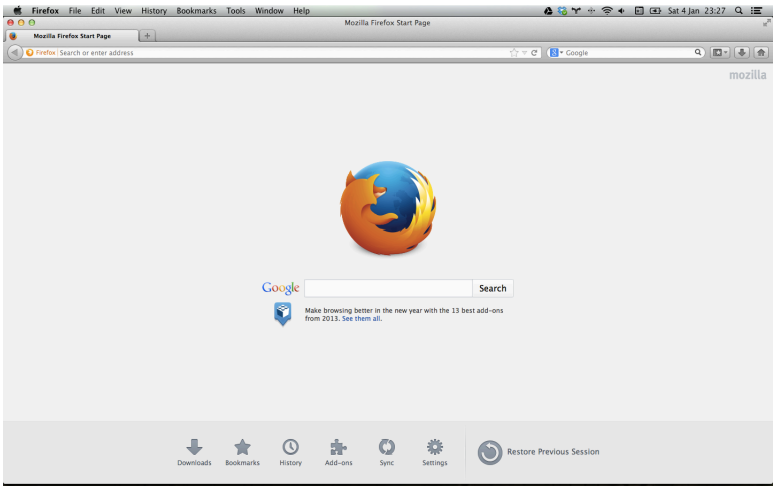


Safari 7

Since Safari (tested with version 7.0) is already installed, all you have to do is to open IRB and type this. If you are not familiar with IRB, see *IRB* chapter.

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :safari
=> #<Selenium::WebDriver::Driver:0x..
f93d5546968bec45e browser=:safari>

> browser.get "http://watir.com"
=> nil
```

Firefox



Firefox 26

To drive Firefox[22] (tested with version 25.0.1), make sure you have it installed. Open our old friend IRB and type this:

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :firefox
=> #<Selenium::WebDriver::Driver:
0x10e1416dd9107ffe browser=:firefox>

> browser.get "http://watir.com"
=> ""
```

Great! We can drive Firefox.

Homebrew

To drive Chrome, you need Homebrew[23]. To install it, paste this into Terminal. You will have to type your password during installation.

[22] http://www.mozilla.org/en-US/firefox/new/
[23] http://brew.sh/

```
$ ruby -e "$(curl -fsSL
https://raw.github.com/mxcl/homebrew/go/install)"
(...)
==> Installation successful!
You should run `brew doctor' *before* you install
anything.
Now type: brew help
```

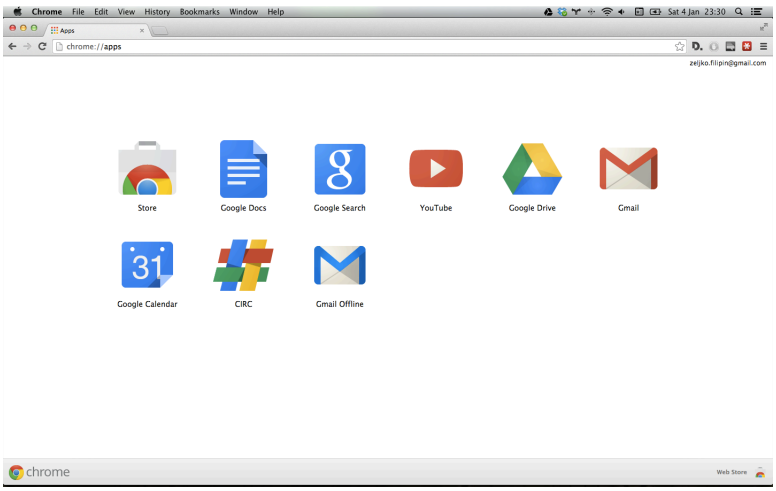
To check if everything is set up correctly, type `brew doctor`:

```
$ brew doctor
Your system is ready to brew.
```

Everything looks good!

(You can thank me later for Homebrew.)

Chrome



Chrome 31

To drive Chrome[24] (tested with version 31), make sure you have it installed.

[24] https://www.google.com/intl/en/chrome/browser/

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :chrome
Selenium::WebDriver::Error::WebDriverError: Unable
to find the chromedriver executable. Please
download the server from
http://code.google.com/p/chromedriver/downloads/
list and place it somewhere on your PATH. More
info at
http://code.google.com/p/selenium/wiki/
ChromeDriver.
(...)
```

Looks like we have to install something called *ChromeDriver executable*. The easiest way to install ChromeDriver is via Homebrew.

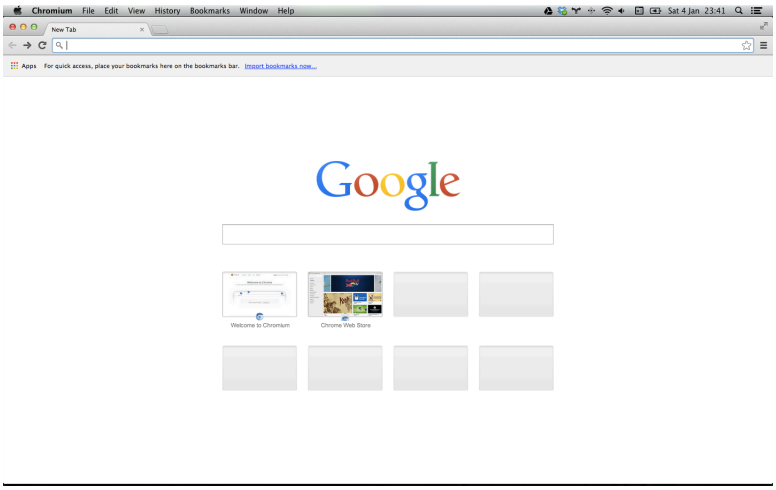
```
$ brew install chromedriver
(...)
```

Let's try again:

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :chrome
=> #<Selenium::WebDriver::Driver:
0xec6568f803e9898 browser=:chrome>

> browser.get "http://watir.com"
=> ""
```

Finally! It works!

Chromium



Chromium 34

Let's try driving Chromium[25] (tested with version 33) too, just for fun. Download zip file from download-chromium.appspot.com[26]. Unzip the file and move `Chromium.app` file to `/Applications` folder. If you did not already install ChromeDriver, see Chrome chapter. To open the browser for the first time you will have to right click it while holding control, then click *Open* from context menu. The next time you can open it in an usual way.

```
$ irb

> require "selenium-webdriver"
=> true

> Selenium::WebDriver::Chrome.path =
"/Applications/Chromium.app/Contents/MacOS/
Chromium"
=> "/Applications/Chromium.app/Contents/MacOS/
Chromium"

> browser = Selenium::WebDriver.for :chrome
=> #<Selenium::WebDriver::Driver:
0x1e35d5faa9511ec6 browser=:chrome>

> browser.get "http://watir.com"
=> nil
```

[25] http://www.chromium.org/

[26] https://download-chromium.appspot.com/

PhantomJS

To drive PhantomJS[27] (tested with version 1.9.2), make sure you have it installed. The easiest way to install it is via Homebrew. (You can thank me now for Homebrew. You are welcome.)

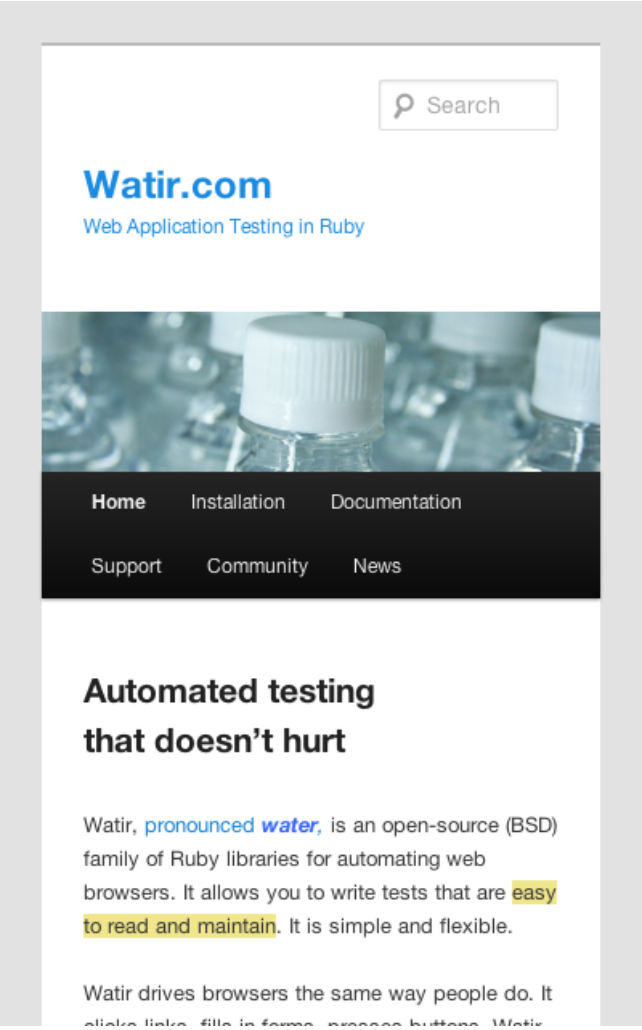
```
$ brew install phantomjs
(...)
```

Let's try driving it:

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :phantomjs
=> #<Selenium::WebDriver::Driver:0x..
fbdc736b89bb162d0 browser=:phantomjs>

> browser.get "http://watir.com"
=> ""

> browser.save_screenshot "phantomjs.png"
=> #<File:phantomjs.png (closed)>
```

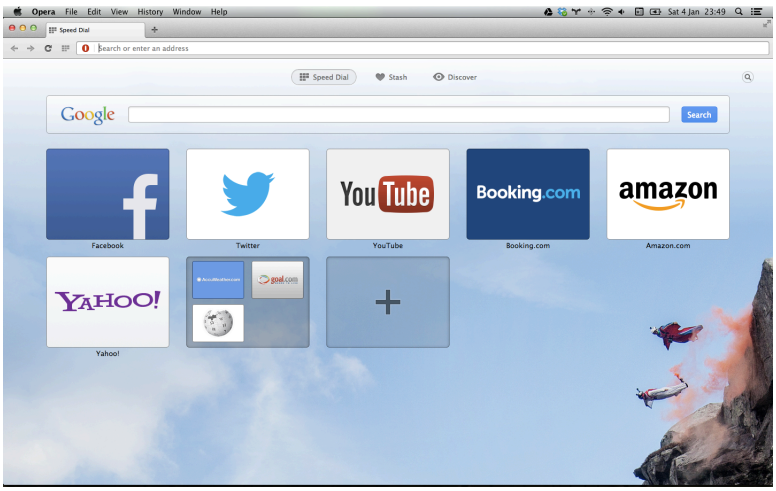
The last command saves screenshot of the page. A screenshot from a headless browser. Nice, right?

[27] http://phantomjs.org/



PhantomJS

Opera



Opera 18

To drive Opera[28] (tested with version 12.16), make sure you have it installed.

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
Selenium::WebDriver::Error::WebDriverError: Unable
to find the Selenium server jar. Please download
the standalone server from
http://code.google.com/p/selenium/downloads/list
and set the SELENIUM_SERVER_JAR environmental
variable to its location. More info at
http://code.google.com/p/selenium/wiki/OperaDriver.
(...)
```

Error message similar to the one when we first tried to open Chrome. The solution is similar too. Install selenium-server-standalone via Homebrew! (If you did not thank me for Homebrew, you can do it now. You are welcome.)

```
$ brew install selenium-server-standalone
(...)
```

Let's try again:

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
```

Install Java

[28] http://www.opera.com/

A popup window will appear saying *To open "java," you need a Java SE 6 runtime. Would you like to install one now?*. Click buttion *Install*, agree with license agreement and Java will install.

Check if Java is installed. Open new Terminal window or tab (it is important to open new window or tab, Terminal will not see Java otherwise) and type `java -version`:

```
$ java -version
java version "1.6.0_65"
Java(TM) SE Runtime Environment
(build 1.6.0_65-b14-462-11M4609)
Java HotSpot(TM) 64-Bit Server VM
(build 20.65-b04-462, mixed mode)
```

The last step is setting `SELENIUM_SERVER_JAR` environmental variable.

If you just want to try driving Opera, typing this into Terminal will do the trick (if you have a newer version of `selenium-server-standalone` file, replace `2.37.0` appropriately):

```
$ export SELENIUM_SERVER_JAR=
/usr/local/opt/selenium-server-standalone/
selenium-server-standalone-2.37.0.jar
```

Let's drive Opera, finally! (Following steps will work only in Terminal tab or window where you have exported `SELENIUM_SERVER_JAR` environment variable.)

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
Selenium::WebDriver::Error::UnknownError: Invalid
service list received:
(...)
(java.lang.IllegalStateException)
(...)
```

If you get above error message, install an older version of Opera. Looks like Selenium can not drive newer versions. The last version that I managed to drive was 12.16. You can get older versions at opera.com/download/guide/?os=mac&list=all[29] or arc.opera.com/pub/opera[30].

Let's try again.

[29] http://www.opera.com/download/guide/?os=mac&list=all

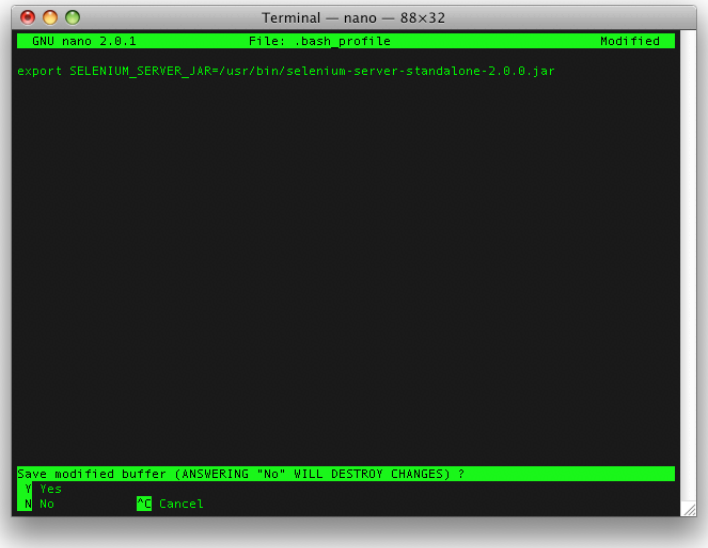
[30] http://arc.opera.com/pub/opera/

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
=> #<Selenium::WebDriver::Driver:0x..
fc28c93dae7536a48 browser=:opera>

> browser.get "http://watir.com"
 => nil
```

Success!

If you plan to drive Opera frequently, you should add `SELENIUM_SERVER_JAR` to `.bash_profile` file. Create (if the file does not exist) or edit `.bash_profile` file in your home folder (`/Users/zeljko/.bash_profile` in my case, or shorter `~/.bash_profile`) with your favorite text editor. Add `export SELENIUM_SERVER_JAR...` line to the file.

This is how to do it with *GNU nano*. Type type `nano ~/.bash_profile`. *GNU nano* text editor will open. Paste (cmd-v, for example) `export SELENIUM_SERVER_JAR...` line. Exit *GNU nano* and save the file with *control+x*. Press *y* when it asks `Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES)?` and press *Enter* when it displays `File Name to Write: .bash_profile` or `Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES)?` (text is different if the file already exists or not).



GNU nano asking should it save changes to `.bash_profile` file

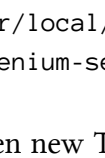
If you have done everything right, GNU nano will close and you will see normal Terminal window. We can check if the line is written to `.bash_profile` file:

```
$ cat ~/.bash_profile
export SELENIUM_SERVER_JAR=
/usr/local/opt/selenium-server-standalone/
selenium-server-standalone-2.37.0.jar
```

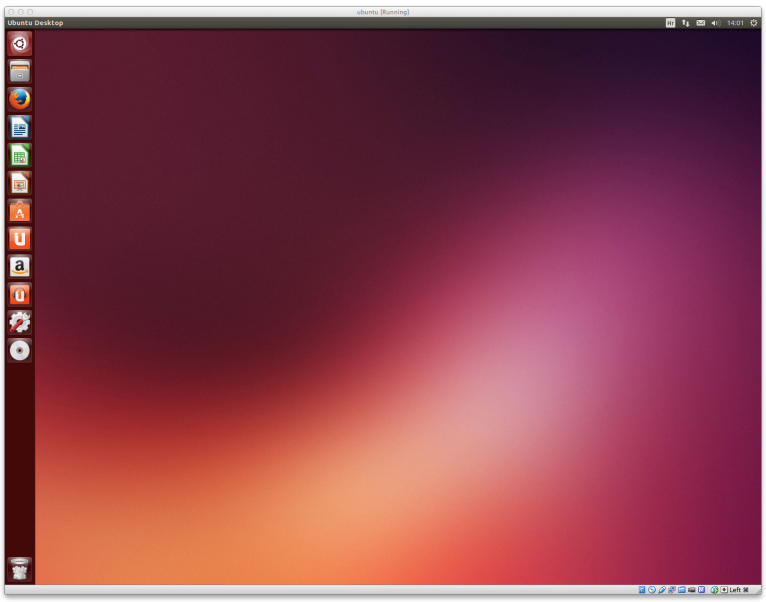
Open new Terminal window or tab (this is important, already opened windows or tabs would not see `SELENIUM_SERVER_JAR` variable). Run the same commands again and everything should just work.

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
=> #<Selenium::WebDriver::Driver:0x..
fc28c93dae7536a48 browser=:opera>

> browser.get "http://watir.com"
 => nil
```

Ubuntu Linux 13.10



You will need internet access if you want to follow examples. All examples are tested with Ubuntu Linux 13.10 32-bit. All browsers are English (US) version.



Ubuntu Linux 13.10

Ruby

Let's see if Ubuntu comes with Ruby installed. Open Terminal and type `ruby -v`:

```
$ ruby -v
The program 'ruby' can be found in the following
packages:
 * ruby1.8
 * ruby1.9.1
Try: sudo apt-get install <selected package>
```

Install it with `sudo apt-get install ruby1.9.1`:

```
$ sudo apt-get install ruby1.9.1
(...)
Setting up ruby1.9.1 (1.9.3.194-8.1ubuntu2.1) ...
(...)
```

Check the version with `ruby -v`:

```
$ ruby -v
ruby 1.9.3p194 (2012-04-20 revision 35410)
[i686-linux]
```

RubyGems

Let's see the version of RubyGems we got with Ruby with `gem -v`:

```
$ gem -v
1.8.23
```

Update RubyGems with `gem update --system`:

```
$ gem update --system
ERROR:  While executing gem ... (RuntimeError)
gem update --system is disabled on Debian, because
it will overwrite the content of the rubygems
Debian package, and might break your Debian system
in subtle ways. The Debian-supported way to update
rubygems is through apt-get, using Debian official
repositories. If you really know what you are
doing, you can still update rubygems by setting
the REALLY_GEM_UPDATE_SYSTEM environment variable,
but please remember that this is completely
unsupported by Debian.
```

Since I do not *really* know what I am doing, I will leave RubyGems at current version and hope everything will work.

selenium-webdriver

Let's try selenium-webdriver gem. Install it with `sudo gem install selenium-webdriver --no-ri --no-rdoc`.

```
$ sudo gem install selenium-webdriver --no-ri
--no-rdoc
(...)
Fetching: ffi-1.9.3.gem (100%)
Building native extensions.  This could take a
while...
ERROR:  Error installing selenium-webdriver:
ERROR: Failed to build gem native extension.
(...)
```

This should fix the problem:

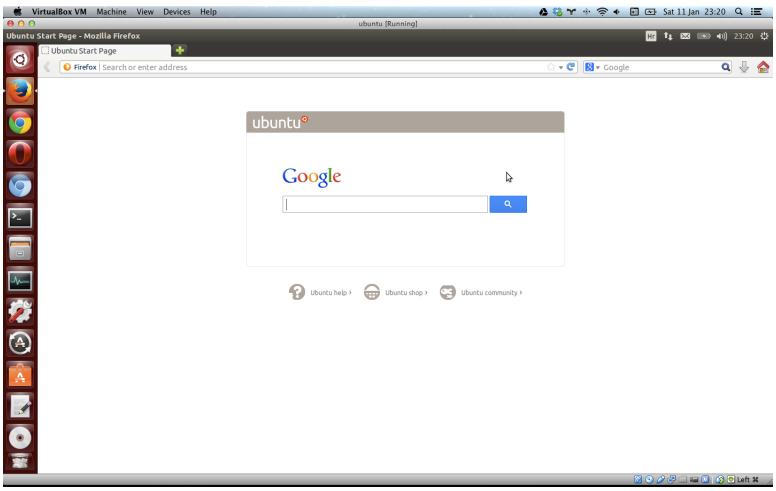
```
$ sudo apt-get install ruby1.9.1-dev
(...)
Setting up ruby1.9.1-dev (1.9.3.194-8.1ubuntu2.1)
```

Let's try installing selenium-webdriver again:

```
$ sudo gem install selenium-webdriver --no-ri
--no-rdoc
(...)
Successfully installed selenium-webdriver-2.37.0
(...)
```

Sucess!

Firefox



Firefox 25

Since Firefox (tested with version 25.0.1) is installed by default on Ubuntu, you do not have to install it.

Let's check if it can drive Firefox:

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :firefox
=> #<Selenium::WebDriver::Driver:0x21564606
browser=:firefox>

> browser.get "http://watir.com"
=> ""
```

No problem here, works just fine.

PhantomJS

To drive PhantomJS[31] (tested with version 1.9.0), install it via `apt-get`:

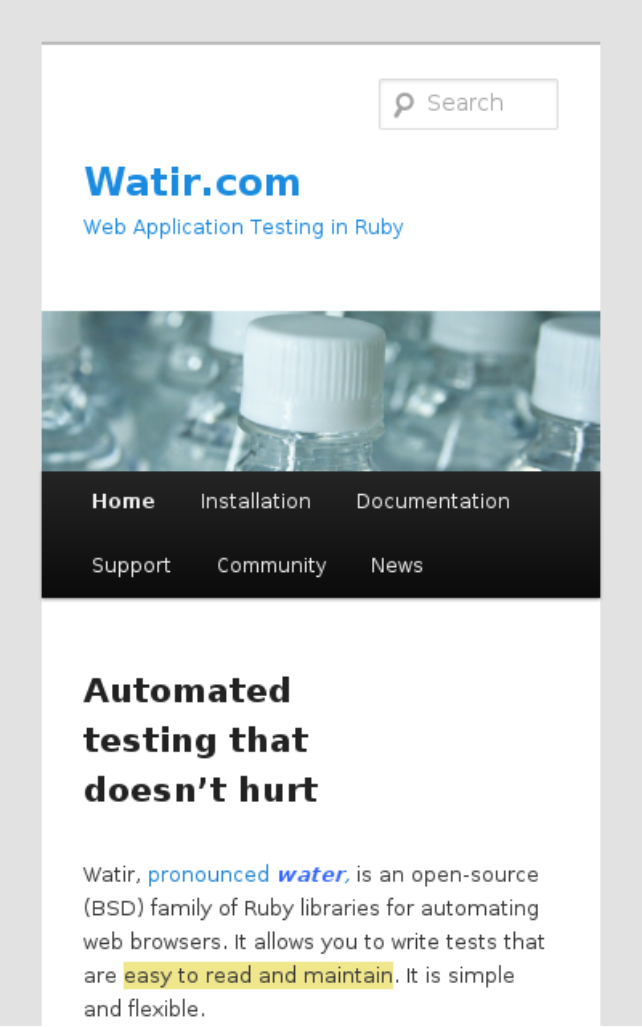
```
$ sudo apt-get install phantomjs
(...)
Setting up phantomjs (1.9.0-1) ...
```

Let's try driving it:

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :phantomjs
=> #<Selenium::WebDriver::Driver:0x..fa5f26f44
browser=:phantomjs>

> browser.get "http://watir.com"
=> {}

> browser.save_screenshot "phantomjs.png"
=> #<File:phantomjs.png (closed)>
```

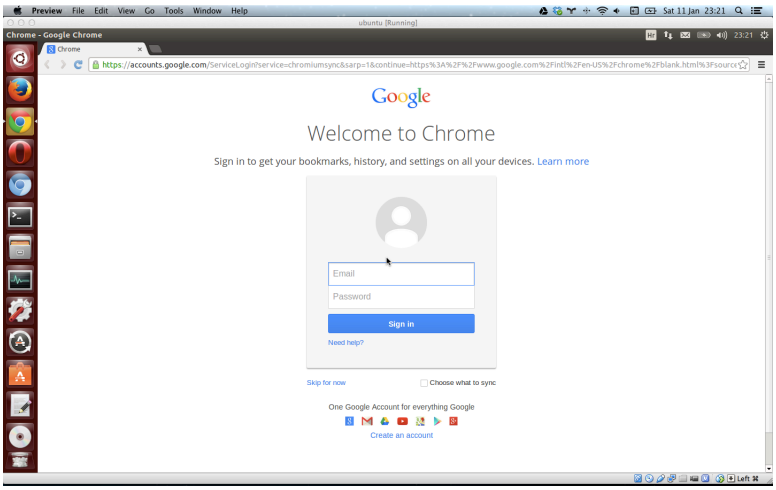
[31] http://phantomjs.org/

The last command saves screenshot of the page. A screenshot from a headless browser. Nice, right?



PhantomJS 1.9.0

Chrome



Chrome 31

Now, lets see if it can really drive Chrome (tested with version 31) too. Ubuntu does not have Chrome installed by default, so you have to install it yourself. Download it from *google.com/chrome*[32].

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :chrome
Selenium::WebDriver::Error::WebDriverError: Unable
to find the chromedriver executable. Please
download the server from
http://code.google.com/p/chromedriver/downloads/
list and place it somewhere on your PATH. More
info at
http://code.google.com/p/selenium/wiki/
ChromeDriver.
(...)
```

Looks like we have to install something called *ChromeDriver executable*. Fortunately, the error message is pretty clear. code.google.com/p/chromedriver/downloads/list[33] will let you know that *ChromeDriver executable* is now located at chromedriver.storage.googleapis.com[34]. Download the latest version of `chromedriver_linux32.zip` or `chromedriver_linux64.zip` (check if you have 32-bit or 64-bit operating system) and unzip it (mouse right click and then *Extract Here*, for

[32] http://www.google.com/chrome
[33] http://code.google.com/p/chromedriver/downloads/list
[34] http://chromedriver.storage.googleapis.com/index.html

example). You will get a file named `chromedriver`. Put it *somewhere on your `PATH`*, as the error message said.

Let's find out what is *on our `PATH`*.

```
$ echo $PATH
/home/zeljko/bin:/usr/local/sbin:/usr/local/bin:
/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

`/usr/bin` looks like a nice place, so let's move `chromedriver` there. You will have to provide your password to move the file there.

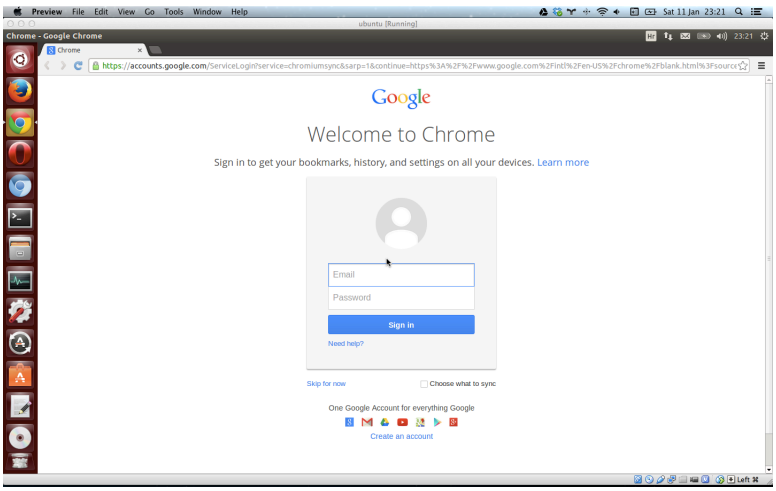
```
$ sudo mv chromedriver /usr/bin
```

Let's drive Chrome, finally:

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :chrome
=> #<Selenium::WebDriver::Driver:0x5b77effe
browser=:chrome>

> browser.get "http://watir.com"
=> nil
```

And it really works!

Chromium



Chromium 30

Let's try driving Chromium[35] (tested with version 30) too, just for fun. Install it with `sudo apt-get install chromium-browser`:

```
$ sudo apt-get install chromium-browser
(...)
Setting up chromium-browser
(30.0.1599.114-0ubuntu0.13.10.2) ...
(...)
```

If you did not already install ChromeDriver, see *Chrome* chapter.

```
$ irb

> require "selenium-webdriver"
=> true

> Selenium::WebDriver::Chrome.path =
"/usr/bin/chromium-browser"
=> "/usr/bin/chromium-browser"

> browser = Selenium::WebDriver.for :chrome
=> #<Selenium::WebDriver::Driver:0x..fb4c9860a
browser=:chrome>

> browser.get "http://watir.com"
=> nil
```

Java

To drive Opera, you will have to install Java first. Let's check if Java is already installed:

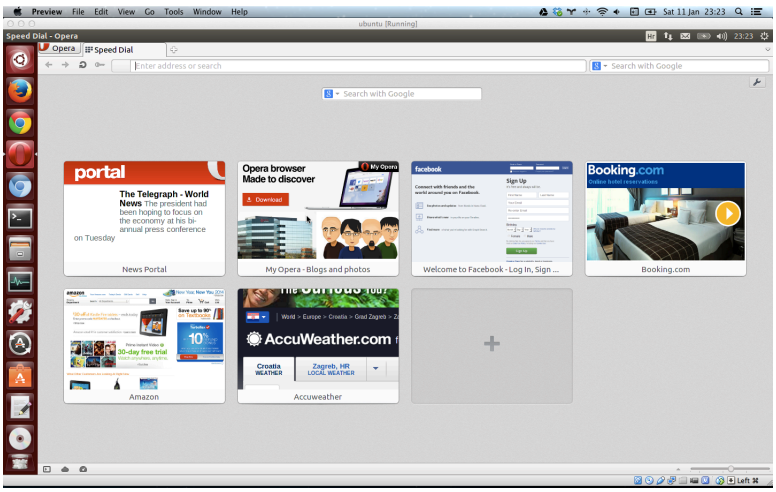
```
$ java -version
The program 'java' can be found in the following
packages:
 * default-jre
 * gcj-4.6-jre-headless
 * gcj-4.7-jre-headless
 * openjdk-7-jre-headless
 * openjdk-6-jre-headless
Try: sudo apt-get install <selected package>
```

Looks like we will have to install Java. Install it with `sudo apt-get install openjdk-7-jre-headless`:

[35] http://www.chromium.org/

```
$ sudo apt-get install openjdk-7-jre-headless
(...)
Setting up openjdk-7-jre-lib (7u25-2.3.12-4ubuntu3)
(...)
```

Opera



Opera 12.16

To drive Opera[36] (tested with version 12.16) make sure you have it installed.

Let's see how it drives Opera. Open our old friend, IRB:

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
Selenium::WebDriver::Error::WebDriverError: Unable
to find the Selenium server jar. Please download
the standalone server from
http://code.google.com/p/selenium/downloads/list
and set the SELENIUM_SERVER_JAR environmental
variable to its location. More info at
http://code.google.com/p/selenium/wiki/OperaDriver.
(...)
```

Error message similar to the one when we first tried to open Chrome. The solution is similar too. We have to download a file, put it somewhere and point a variable to it. Do not worry, it sounds more complicated than it really is. Fortunately again, the error message says it all. Go to

[36] http://www.opera.com/

code.google.com/p/selenium/downloads/list[37] and download `selenium-server-standalone-2.39.0.jar` (or newer version, the description should be *Use this if you want to use the Selenium RC or Remote WebDriver or use Grid 2 without needing any additional dependencies*).

The last step is setting `SELENIUM_SERVER_JAR` environmental variable. If you just want to try driving Opera, typing this into Terminal will do the trick (assuming that the file is located in Downloads folder, if you have a newer version of `selenium-server-standalone` file, replace `2.39.0` appropriately):

```
$ export SELENIUM_SERVER_JAR=
~/Downloads/selenium-server-standalone-2.39.0.jar
```

Let's drive Opera, finally! (Following steps will work only in Terminal tab or window where you have exported `SELENIUM_SERVER_JAR` environment variable.)

```
$ irb

> require "selenium-webdriver"
=> true

> browser = Selenium::WebDriver.for :opera
=> #<Selenium::WebDriver::Driver:0x..fb4bb92ce
browser=:opera>

> browser.get "http://watir.com"
=> nil
```

If you plan to drive Opera frequently, you should add `SELENIUM_SERVER_JAR` to `.bashrc` file. Create (if the file does not exist) or edit `.bashrc` file in your home folder (`/home/z/.bashrc` in my case, or shorter `~/.bashrc`) with your favorite text editor. Add `export SELENIUM_SERVER_JAR...` line to the file.

```
$ nano ~/.bashrc
```

Save the file (ctrl+x) and close all Terminal windows. Open Terminal again. To check if the variable is set, try `printenv | grep SELENIUM`:

```
$ printenv | grep SELENIUM
SELENIUM_SERVER_JAR=/home/zeljko/bin/
selenium-server-standalone-2.39.0.jar
```

Looks good to me!

[37] http://code.google.com/p/selenium/downloads/list

Quick Start

You will need internet access if you want to follow examples in this chapter. If you do not have Ruby, RubyGems, Selenium and Firefox installed, please see *Installation* chapter. If you are not familiar with Command-line interface or IRB, see *Command-line interface* and *IRB* chapters.

To start IRB, just type `irb` in command line. You will see something like this:

```
$ irb
>
```

Now you can enter any Ruby command and you will immediately get a result. We will start with telling Ruby that we want to use selenium-webdriver gem with `require "selenium-webdriver"`:

You should see something like this:

```
> require "selenium-webdriver"
=> true
```

Every Ruby command returns something. You should get `=> true` after `require "selenium-webdriver"`. There are two parts in the returned line. The first one is `=>`. It looks like an arrow. It means `Ruby returned this`. The second part is `true`, the thing that is actually returned. When `true` is returned, it usually means that everything is fine. Unless I say differently, just ignore what is returned, for now.

Open Firefox.

```
> browser = Selenium::WebDriver.for :firefox
=> #<Selenium::WebDriver::Driver:...
browser=:firefox>
```

As I said earlier, you can ignore `#<Selenium::WebDriver::Driver:...>`. Opening Firefox returned the browser as an object, and this is textual representation of the object.

Just opening a browser is not so useful. Let's open *google.com*. I would suggest that you literally follow the example, and then try a few sites yourself.

So, go to *google.com*:

```
> browser.get "https://www.google.com/"
=> ""
```

And *google.com* opens. (Since I am in Croatia, *google.hr* opened. If you are not in the US, some other Google site could open.)

Controlling the browser is really useful but, as I am sure you already know, there is more to testing than just performing the actions. You have to check what happened after the action. What happens when I enter a URL in browser address bar, when I click a link or a button, when I enter some text in a text field or select something from select box...?

This is the first time we will perform a check. It is also the first time we will take a look what Ruby returns after the command. Let's check if the browser really opened *google.com*.

```
> browser.current_url
=> "https://www.google.hr/"
```

It really works! Ruby returned a string (the thing in double quotes) that contains the text from the browser address bar.

It's time to click on a link. It is easy to explicitly say which link to click on. Right now I want to click on a link with the text *Google.com*. If your browser already opened *google.com*, ignore this step.

```
> browser.find_element(link_text: "Google.com").
click
=> "ok"
```

And *google.com* opens. Now that all of us are on literally on the same page, let's click another link:

```
> browser.find_element(link_text: "Images").click
=> "ok"
```

This time, let's check the page title.

```
> browser.title
=> "Google Images"
```

We got back the string with the page title.

Let's search for something. This will enter *book* in search text field:

```
> browser.find_element(name: "q").send_keys "book"
=> ""
```

Maybe you are wondering how I knew the text field had the value of `name` attribute set to `q`. (I am talking about `name: "q"`.) If you do not know how to inspect pages, read on. It will be explained in *Browser Developer Tools* chapter.

Now, click the search button:

```
> browser.find_element(name: "btnG").click
=> "ok"
```

Page with search results will open. Let's check how many images are on the page.

```
> browser.find_elements(tag_name: "img").size
=> 126
```

You might get a different number, it does not have to be 126. And finally, let's close the browser.

```
> browser.close
=> ""
```

Well, that was a lot of fun. But you do not want to type into IRB all the time. You want to run the tests and do something else while they run. As for almost everything else in Ruby, there is a simple solution. Paste all code you have entered in IRB in a text file, and save it with *rb* extension. IRB is used only for development or debugging, so do not paste `irb` as the first line of the file. The file should look like this:

```
require "selenium-webdriver"
browser = Selenium::WebDriver.for :firefox
browser.get "https://www.google.com/"
browser.current_url
browser.find_element(link_text: "Google.com").click
browser.find_element(link_text: "Images").click
browser.title
browser.find_element(name: "q").send_keys "book"
browser.find_element(name: "btnG").click
browser.find_elements(tag_name: "img").size
browser.close
```

You can use any text editor to edit the file. I use RubyMine[38] or Sublime Text[39].

Save the file as `quick_start.rb`. If IRB is still running in your command line, press `ctrl+d` to return to normal command prompt, or open a new command prompt. (To exit from IRB to normal command line, instead of pressing `ctrl+d`, you can also type `quit` or `exit`.) You should remove clicking *Google.com* link if Firefox opens it automatically for you.

To run the file, navigate in command prompt to the folder where you have saved it and run it:

[38] http://www.jetbrains.com/ruby/

[39] http://www.sublimetext.com/3

```
$ ruby quick_start.rb
```

Smile while the browser clicks around.

What is the output in the command prompt? Nothing? Yes, nothing. IRB displays values that Ruby returns, but when you execute Ruby file from the command line, it does not display the values Ruby returns. You have to explicitly say to Ruby that you want them displayed. It is as easy as adding `p` in front of the command. Add `p` in front of three lines. Modify the script to look like this. You can add `p` in front of every command, but you really do not care about what some commands return:

```
require "selenium-webdriver"
browser = Selenium::WebDriver.for :firefox
browser.get "https://www.google.com/"
p browser.current_url
browser.find_element(link_text: "Google.com").click
browser.find_element(link_text: "Images").click
p browser.title
browser.find_element(name: "q").send_keys "book"
browser.find_element(name: "btnG").click
p browser.find_elements(tag_name: "img").size
browser.close
```

Run the script. This time the output should look like this:

```
$ ruby quick_start.rb
"https://www.google.hr/"
"Google Images"
126
```

Later I will show you how to make cool looking reports.

Getting Help

TODO

How to get help might be most important thing to learn. In general good resources are Stack Overflow[40], Google Groups[41], IRC[42] and project web sites.

It might seem almost a joke that project web sites are mentioned, but all the time I see people asking for help without doing even the basic research. Also, common pattern is asking for help in the wrong place. You will not get good support asking for help with Selenium on a general purpose Ruby mailing list, or asking a basic Ruby question at a Selenium mailing list. You have to learn enough Ruby to know where the language (Ruby) ends and where a library (like Selenium) starts.

It is really important that you learn where you can get more information. A good start is selenium-webdriver[43] gem page at RubyGems.org[44]. You will find links to it's homepage[45], source code[46], API documentation[47], wiki[48] and bug tracker[49] there. Another good API documentation is automatically generated at selenium-webdriver[50] section of rdoc.info[51] site. Looks like rdoc.info is more up to date than the official API documentation. At the moment, you should focus on API documentation and wiki. The wiki says that Selenium::WebDriver::Driver[52] and Selenium::WebDriver::Element[53] are the two main classes.

[40] http://stackoverflow.com/
[41] https://groups.google.com
[42] https://en.wikipedia.org/wiki/Internet_Relay_Chat
[43] https://rubygems.org/gems/selenium-webdriver
[44] https://rubygems.org/
[45] http://selenium.googlecode.com/
[46] http://code.google.com/p/selenium/source/list
[47] http://selenium.googlecode.com/svn/trunk/docs/api/rb/index.html
[48] http://code.google.com/p/selenium/wiki/RubyBindings
[49] http://code.google.com/p/selenium/issues/list
[50] http://rdoc.info/gems/selenium-webdriver/
[51] http://rdoc.info/
[52] http://selenium.googlecode.com/svn/trunk/docs/api/rb/Selenium/WebDriver/Driver.html
[53] http://selenium.googlecode.com/svn/trunk/docs/api/rb/Selenium/WebDriver/Element.html

Driver

You will need internet access if you want to follow examples in this chapter. All examples in this chapter are tried on Mac OS X 10.8.5, Firefox 27.0.1, Ruby 2.1.0p0 and selenium-webdriver 2.40.0 but everything should work on all supported platforms. If you are not familiar with Command-line interface or IRB, see *Command-line interface* and *IRB* chapters.

In this chapter you will learn how to control the browser using Selenium. In Quick Start chapter we have touched two important part of Selenium API, driver and element. In this chapter we will focus on the driver. The chapter will not cover the entire driver API, for more information see Selenium::WebDriver::Driver[54] API documentation.

The first thing you have to do is to open the browser. According to the documentation, currently supported browsers are Firefox, Internet Explorer, Chrome, Android, iPhone, Opera, PhantomJS, Safari and remote browser. Let's focus on desktop drivers for now. See *Mobile* chapter on how to drive mobile browsers and *Selenium in the Cloud* chapter on how to driver remote browsers. How to set up everything is explained in detail in *Installation* chapter.

Browser	Symbol	Shortcut
Chrome	:chrome	
Firefox	:firefox	:ff
Internet Explorer	:internet_explorer	:ie
Opera	:opera	
PhantomJS	:phantomjs	
Safari	:safari	

How to start a browser

Open Firefox:

[54] http://rdoc.info/gems/selenium-webdriver/Selenium/WebDriver/Driver

```
$ irb

> require "selenium-webdriver"
 => true

> browser = Selenium::WebDriver.for :firefox
=> #<Selenium::WebDriver::Driver:0x..
f8698791d2bff9778 browser=:firefox>
```

or

```
> browser = Selenium::WebDriver.for :ff
=> #<Selenium::WebDriver::Driver:...
browser=:firefox>
```

If you want to drive a different browser, just replace `:firefox` with another symbol.

Let's try a few interesting methods. `get` opens a page:

```
> browser.get "http://google.com"
=> ""
```

`current_url` returns page URL:

```
> browser.current_url
=> "https://www.google.hr/"
```

`title` returns page title:

```
> browser.title
=> "Google"
```

`close` closes the current window or the entire browser if there is only one window open.

When you are done with the browser, close it:

```
> browser.close
=> ""
```

If you want to close multiple browser windows at once, use `quit`:

```
> browser.quit
=> nil
```

Create a Ruby file from the above IRB session and save it as `driver.rb`. Of course, add a `p` in front of a few commands, so the script outputs something.

```
require "selenium-webdriver"
browser = Selenium::WebDriver.for :firefox
browser.get "http://google.com"
p browser.current_url
p browser.title
browser.close
browser.quit
```

Run the file:

```
$ ruby driver.rb
"https://www.google.hr/"
"Google"
```

Element

You will need internet access if you want to follow examples in this chapter. All examples in this chapter are tried on Mac OS X 10.8.5, Firefox 27.0.1, Ruby 2.1.0p0 and selenium-webdriver 2.40.0 but everything should work on all supported platforms. If you are not familiar with Command-line interface or IRB, see *Command-line interface* and *IRB* chapters.

After you had a taste of Selenium in *Getting Started* chapter, it is time to take a closer look at the Selenium API. This chapter will get you introduced to the API but it will not cover it entirely. For more information see Selenium::WebDriver::Element[55] API documentation.

Let's start Firefox and open Google home page:

```
$ irb

> require "selenium-webdriver"
 => true

> browser = Selenium::WebDriver.for :firefox
=> #<Selenium::WebDriver::Driver:0x..
f8698791d2bff9778 browser=:firefox>

> browser.get "http://google.com"
=> ""
```

Most of the time you will be dealing with page elements, so let's see how to do that. First, you need to find the element on the page, then you usually need to do something with it, like entering text or clicking it. To find the element, use `find_element` method.

```
> browser.find_element(name: "q")
=> #<Selenium::WebDriver::Element:...
   id="{25201324-ac0c-8e40-9766-c35aa5b54786}">
```

There are many options available to find an element.

[55] http://rdoc.info/gems/selenium-webdriver/Selenium/WebDriver/Element

How	Symbol	Shortcut
class name	:class_name	:class
css selector	:css	
id	:id	
link text	:link_text	:link
name	:name	
partial link text	:partial_link_text	
tag name	:tag_name	
xpath	:xpath	

How to find an element

Looks like we have found the element, but we did not do anything with it yet. Since the element we have found is a text field, let's enter some text into it:

```
> browser.find_element(name: "q").send_keys "watir"
=> ""
```

We can also clear the text field:

```
> browser.find_element(name: "q").clear
=> ""
```

There is a shortcut if you want to find an element via it's `id`. Just provide the `id`:

```
> browser["gbqfq"]
=> #<Selenium::WebDriver::Element:...
   id="{fca96529-8bc6-bf4f-8e78-376f037c351a}">
```

To get the value of any element attribute, use `attribute`. Try a few attributes:

```
> browser["gbqfq"].attribute(:name)
=> "q"

> browser["gbqfq"].attribute(:class)
=> "gbqfif"

> browser["gbqfq"].attribute(:type)
=> "text"

> browser["gbqfq"].attribute(:autocomplete)
=> "off"

> browser["gbqfq"].attribute(:style)
```

```
=> "border: medium none; padding: 0px; margin: ...

> browser["gbqfq"].attribute(:outerHTML)
=> "<input spellcheck=\"false\" dir=\"ltr\" ...
```

Create a Ruby file from the above IRB session and save it as `element.rb`. Of course, add a `p` in front of a few commands, so the script outputs something.

```
require "selenium-webdriver"
browser = Selenium::WebDriver.for :firefox
browser.get "http://google.com"
browser.find_element(name: "q")
browser.find_element(name: "q").send_keys "watir"
browser.find_element(name: "q").clear
browser["gbqfq"]
p browser["gbqfq"].attribute(:name)
p browser["gbqfq"].attribute(:class)
p browser["gbqfq"].attribute(:type)
p browser["gbqfq"].attribute(:autocomplete)
p browser["gbqfq"].attribute(:style)
p browser["gbqfq"].attribute(:outerHTML)
```

Run the file:

```
$ ruby element.rb
"q"
"gbqfif"
"text"
"off"
"border: medium none; padding: 0px; margin: 0px;...
"<input spellcheck=\"false\" dir=\"ltr\" style=...
```

Nesting

TODO

How and why to use nesting when working with page elements.

Collections

TODO

How to work with collections of page elements.

Frames

TODO

Frames have a special treatment.

Pop Ups

TODO

Pop up windows are causing a lot of trouble for people.

Headless

I have noticed a lot of confusion about headless testing, but there is really nothing special there. The confusion may be caused by the term *headless* being a bit vague. In the context of driving a browser, *headless* means you can drive a real browser, but without seeing anything on the machine. It can be useful for running tests on a headless machine, or on a desktop machine. For example, if you want to run tests on a headless machine as part of continuous integration, or if you want to run tests on your desktop machine without browsers opening and closing on the screen all the time, while you are doing something else.

There are two ways to run browser in a headless mode that I am aware of. One is to use a headless browser like PhantomJS[56]. Another option is using Xvfb[57] (X virtual framebuffer) and Headless[58] gem. Please notice that Xvfb works only on Linux.

The advantage of running tests in PhantomJS is that it is supported on all major operating systems. The browser is pretty good, it has the same Selenium API as all other browsers, so the vast majority of the tests developed using another browser will just work. It also has a pretty good JavaScript engine, so even JavaScript heavy pages should work fine. The browser is also faster than other browsers. More speed is always good, but speed improvement depends on a lot of things, so sometimes you will see a lot of improvement, and sometimes just a few percent.

The disadvantage is that the users of your web application are not using PhantomJS to access it, they are using one of the major browsers. Sometimes you will have to tweak the tests or the application to get all tests to run fine. It is also harder to debug failures, since the browser is headless. Fortunately, you can take screen shots and HTML of the page (the entire page or just the relevant part).

The advantage of using Xvfb is that it works with any browser that Selenium can drive. You can delevop tests using your favorite browser and then run them in headless mode with no modifications. The disadvantage is that is it somewhat slower than PhantomJS (but not a lot), and it works only on Linux. Let me repeat that, Xvfb does not work on Windows or Mac OS.

PhantomJS

You will need internet access if you want to follow examples in this chapter. All examples in this chapter are tried on Mac OS X 10.8.5, PhantomJS 1.9.7, Ruby 2.1.1p76 and selenium-webdriver 2.40.0 but everything should work on all supported platforms.

If you do not have Ruby, Selenium and PhantomJS already installed, please see *Installation* chapter. If you are not familiar with Selenium API, please see *Quick Start*, *Driver* and *Element* chapters.

[56] http://phantomjs.org/

[57] https://en.wikipedia.org/wiki/Xvfb

[58] https://github.com/leonid-shevtsov/headless

All you have to do to drive PhantomJS is to let Selenium know that you want to drive it:

```
$ irb

> require "selenium-webdriver"
 => true

> browser = Selenium::WebDriver.for :phantomjs
=> #<Selenium::WebDriver::Driver:...
browser=:phantomjs>
```

The rest of the API is the same as for any other browser. For example, go to a page:

```
> browser.get "http://google.com"
=> {}
```

Get it's URL and title:

```
> browser.current_url
=> "http://www.google.hr/"

> browser.title
=> "Google"
```

Enter text into the text field and then clear the text field:

```
> browser.find_element(name: "q").send_keys "watir"
=> nil

> browser.find_element(name: "q").clear
=> nil
```

Play with element attributes:

```
> browser.find_element(name: "q").attribute(:name)
=> "q"

> browser.find_element(name: "q").attribute(:class)
=> "lst tiah"

> browser.find_element(name: "q").attribute(:type)
=> "text"

> browser.find_element(name: "q").
attribute(:autocomplete)
=> "off"
```

When driving Firefox or any of the usual browsers, you are able to see how to page looks like. With PhantomJS you do not see anything, so it is really important to know how to debug problems. Two features are really useful, taking screenshots and getting page HTML.

To take a screenshot, use `save_screenshot` method:

```
> browser.save_screenshot "phantomjs.png"
=> #<File:phantomjs.png (closed)>
```

To get page HTML, use

```
> browser.page_source
=> "<!DOCTYPE html><html itemscope=\"\"
itemtype=\"http://schema.org/WebPage\"><head><meta
itemprop=\"image\"
content=\"/images/google_favicon_128.png\">
<title>Google</title><script>
...
</script></body></html>"
```

To get HTML of just a part of the page, ask for `outerHTML` attribute:

```
> browser.find_element(name: "q").
attribute(:outerHTML)
=> "<input autocomplete=\"off\" class=\"lst tiah\"
...
```

At the end, close the browser:

```
> browser.quit
=> nil
```

Create a Ruby file from the above IRB session and save it as `headless_phantomjs.rb`. Of course, add a `p` in front of a few commands, so the script outputs something.

```
require "selenium-webdriver"
browser = Selenium::WebDriver.for :phantomjs
browser.get "http://google.com"
p browser.current_url
p browser.title
browser.find_element(name: "q").send_keys "watir"
browser.find_element(name: "q").clear
p browser.find_element(name: "q").attribute(:name)
p browser.find_element(name: "q").attribute(:class)
p browser.find_element(name: "q").attribute(:type)
```

```
11 p browser.find_element(name: "q").
12   attribute(:autocomplete)
13 browser.save_screenshot "phantomjs.png"
14 p browser.page_source
15 p browser.find_element(name: "q").
16   attribute(:outerHTML)
17 browser.quit
```

Run the file:

```
1 $ ruby headless_phantomjs.rb
2 "http://www.google.hr/"
3 "Google"
4 "q"
5 "lst tiah"
6 "text"
7 "off"
8 "<!DOCTYPE html><html itemscope=\"\" itemtype=...
9 "<input autocomplete=\"off\" class=\"lst tiah\"...
```

Xvfb

You will need internet access if you want to follow examples in this chapter. All examples in this chapter are tried on Ubuntu Linux 13.10, Firefox 27.0.1, Ruby 2.1.1p76 and selenium-webdriver 2.40.0 but everything should work on all supported platforms.

Xvfb works only on Linux. It does not work on Windows or Mac OS.

If you do not have Firefox, Ruby or Selenium installed, see *Installation* chapter. If you are not familiar with Selenium API, please see *Quick Start*, *Driver* and *Element* chapters.

Without getting into a lot of technical detail, Xvfb[59] (X virtual framebuffer) is a piece of software that makes is possible to run browsers (and other applications) in a headless mode.

Install Xvfb via `apt-get`:

```
1 $ sudo apt-get install xvfb
2 ...
```

Then install `headless` Ruby gem:

[59] https://en.wikipedia.org/wiki/Xvfb

```
$ gem install headless --no-ri --no-rdoc
...
```

“To sudo or not to sudo, that is the question...”

> On the machine I was using while writing this chapter, Ruby was installed via RVM (Ruby Version Manager). If you are using Ruby that was preinstalled on the computer (as it is on a Mac OS X) or if you have installed Ruby via `apt-get` (on Ubuntu) then you have to add `sudo` in front of `gem`:
>
> ```
> $ sudo gem install headless --no-ri --no-rdoc
> ...
> ```

There are two modes of using *headless* gem, block and object. Block mode will automatically start and destroy headless session. In object mode, you have to explicitly start and destroy the session.

This is an example of block mode (using IRB):

```
$ irb

> require "headless"
=> true

> require "selenium-webdriver"
=> true

> Headless.ly do
>   browser = Selenium::WebDriver.for :firefox
>   browser.get "http://google.com"
>   browser.title
> end
=> "Google"
```

This is an example of object mode (using IRB):

```
$ irb

> require "headless"
=> true

> require "selenium-webdriver"
=> true

> headless = Headless.new
=> #<Headless:0x9e957d8 @display=99,
@autopick_display=true, @reuse_display=true,
@dimensions="1280x1024x24",
@video_capture_options={}, @destroy_at_exit=true>

> headless.start
=> #<Proc:...>

> browser = Selenium::WebDriver.for :firefox
=> #<Selenium::WebDriver::Driver:...
 browser=:firefox>

> browser.get "http://google.com"
=> ""

> browser.title
=> "Google"

> headless.destroy
=> ["/tmp/.X99-lock"]
```

Of course, block and object mode can be used in Ruby files, not just in IRB. Save the following text as `headless_block_mode.rb` file.

```
require "headless"
require "selenium-webdriver"

Headless.ly do
  browser = Selenium::WebDriver.for :firefox
  browser.get "http://google.com"
  p browser.title
end
```

Run the file:

```
$ ruby headless_block_mode.rb
"Google"
```

Save the following text as `headless_object_mode.rb` file.

```
require "headless"
require "selenium-webdriver"

headless = Headless.new
headless.start

browser = Selenium::WebDriver.for :firefox
browser.get "http://google.com"
p browser.title

headless.destroy
```

Run the file:

```
$ ruby headless_object_mode.rb
"Google"
```

As usual, you can take screenshots using Selenium API while running tests, even in headless mode. If you do not know how to do that, see *Driver* chapter.

Xvfb has it's own screenshots and video recording API, but it is beyond the scope of this book to cover it. For more information see documentation for headless[60] gem.

[60] https://github.com/leonid-shevtsov/headless

Selenium in the Cloud

TODO

In addition to being able to drive browsers on your machine, Selenium can also drive browsers on remote machines. This chapter will let you know how to do that using services like Sauce Labs[61], TestingBot[62] and BrowserStack[63].

[61] https://saucelabs.com/

[62] http://testingbot.com/

[63] http://www.browserstack.com/

Mobile

TODO

Android and iPhone drivers will be deprecated really soon, but fortunately there are good alternatives. See Android and iOS Support[64] blog post at Official Selenium Blog[65] for more detail. In this chapter you will learn how to test mobile browsers with tools like Selendroid[66], ios-driver[67] and Appium[68].

[64] http://seleniumhq.wordpress.com/2013/12/24/android-and-ios-support/
[65] http://seleniumhq.wordpress.com/
[66] http://selendroid.io/
[67] http://ios-driver.github.io/ios-driver/
[68] http://appium.io/

Alternative APIs

TODO

In addition to Selenium WebDriver API, there are other APIs that you can use, like Capybara[69], Webrat[70] and Watir[71].

[69] https://github.com/jnicklas/capybara
[70] https://github.com/brynary/webrat
[71] http://watir.com/

Page Object Pattern

TODO

Page[72] Object[73] pattern is the best way to organize your Selenium code. This chapter will show you how you can create a simple implementation of the pattern and how to use page-object gem[74].

[72] https://code.google.com/p/selenium/wiki/PageObjects
[73] http://martinfowler.com/bliki/PageObject.html
[74] https://github.com/cheezy/page-object

Recorders

TODO

Test Frameworks

TODO

While writing tests, you will need a way to set up the environment before the test, clean up after the test, create reports after all tests are executed and similar tasks. There is no need to implement that functionality yourself because Ruby has a lot of good and mature test frameworks. Two really popular ones are Cucumber[75] and RSpec[76].

[75] http://cukes.info/
[76] http://rspec.info/

Ruby Tools

TODO

You will be writing a lot of Ruby code. It is beyond scope of this book to teach Ruby, so if you are not familiar with it, you should read a book or two[77] about it. You should also get familiar with Ruby tools like IRB[78] (Interactive Ruby Shell), RubyGems[79], Bundler[80], RDoc[81], RVM[82] (Ruby Version Manager) and The Ruby Toolbox[83].

IRB (Interactive Ruby Shell)

You do need internet access if you want to follow examples in this chapter. All examples in this chapter are tried on Mac OS X 10.8.5 and Ruby 2.1.1p76 but everything should work on all supported platforms.

If you are familiar with Ruby, I am sure you already think IRB[84] (Interactive Ruby Shell) is one of the greatest tools for learning a new Ruby library. If you are new to Ruby, you are probably thinking: *What is this IRB thing?* IRB is *Interactive Ruby Shell*. We will not get into technical details here, all you need to know is that it is a shell that executes Ruby commands. Every command executes immediately and returns something. Sometimes you will be interested in the return value, the rest of the time you can ignore it.

IRB is a command-line interface (CLI) tool. If you are not familiar with CLI, see *Command-line interface* chapter.

To start IRB, type `irb` in command line. On Mac and Linux, it should look like this:

```
$ irb
2.1.1 :001 >
```

On Windows, it should look like this:

[77] https://www.ruby-lang.org/en/documentation/
[78] https://en.wikipedia.org/wiki/Interactive_Ruby_Shell
[79] http://rubygems.org/
[80] http://bundler.io
[81] http://rdoc.info/
[82] http://rvm.io
[83] https://www.ruby-toolbox.com
[84] https://en.wikipedia.org/wiki/Interactive_Ruby_Shell

```
>irb
DL is deprecated, please use Fiddle
irb(main):001:0>
```

If you see something slightly different, do now worry, if the examples from this chapter work fine. Let's take a look at the output. Mac and Linux have `2.1.1 :001 >` as prompt and Windows has `irb(main):001:0>`. On Mac/Linux, `2.1.1` part is Ruby version. Windows does not display it. Windows has `irb(main)`, so you do not forget you are using IRB. Windows also displays `DL is deprecated, please use Fiddle` warning message. Feel free to ignore it.

`:001` on Mac/Linux and `:001:0` on Windows is line number. If you just press enter/return (depending on your keyboard), you should see something like this.

Mac/Linux:

```
$ irb
2.1.1 :001 >
2.1.1 :002 >
```

Windows:

```
>irb
DL is deprecated, please use Fiddle
irb(main):001:0>
irb(main):002:0*
```

Since the part before the `>` or `*` is not important for us at the moment, we will ignore it in the rest of the book. To differentiate between regular CLI and IRB, the book will use `$` for the regular CLI and `>` for IRB. Let me explain that with a couple of examples.

Executing `ls` command in Mac/Linux CLI:

```
$ ls
Book.txt Gemfile.lock README.md VERSION images lib
misc Gemfile LICENSE.md Subset.txt epub_images
installation main old
```

Executing `1+1` in IRB:

```
$ irb

> 1+1
=> 2
```

The above IRB example shows that Ruby knows how to calculate sum of two numbers. You can also ask it to output something:

```
> p "wow much output so hip"
"wow much output so hip"
=> "wow much output so hip"
```

After outputting the string, Ruby returned it. You can ignore it for now. All you need to know is that it did not complain. Let's try something that will fail:

```
> doge
NameError: undefined local variable or method
`doge' for main:Object
from (irb):7
from /Users/z/.rvm/rubies/ruby-2.1.1/bin/irb:11:in
`<main>'
```

Looks like Ruby does not know about Doge meme[85]. Important thing here is to notice how an error message looks like. Let's do something useful now, like nicely formatting output. First, we will create a `doge` variable:

```
> doge = {dogecoin: "To the moon!",
hello: "This is Doge", wow: "much addon",
so: "internet"}
=> {:dogecoin=>"To the moon!",
:hello=>"This is Doge", :wow=>"much addon",
:so=>"internet"}
```

Then we will try to output is nicely with `pp` library:

```
> pp doge
NoMethodError: undefined method `pp' for
main:Object
from (irb):9
from ...irb:11:in `<main>'
```

Looks like Ruby does not know about `pp` too. All you need to do to fix the problem is to require the library:

```
> require "pp"
=> true
```

Let's output `doge`, finally:

[85] http://knowyourmeme.com/memes/doge

```
> pp doge
{:dogecoin=>"To the moon!",
:hello=>"This is Doge",
:wow=>"much addon",
:so=>"internet"}
=> {:dogecoin=>"To the moon!",
:hello=>"This is Doge", :wow=>"much addon",
:so=>"internet"}
```

PP[86] is a pretty-printer for Ruby objects. It is really useful for outputting big objects.

IRB is not the only Ruby shell. There is irb Alternatives[87] category at The Ruby Toolbox[88]. The most popular alternative tool is Pry[89], but there is another one called ripl[90].

RVM (Ruby Version Manager)

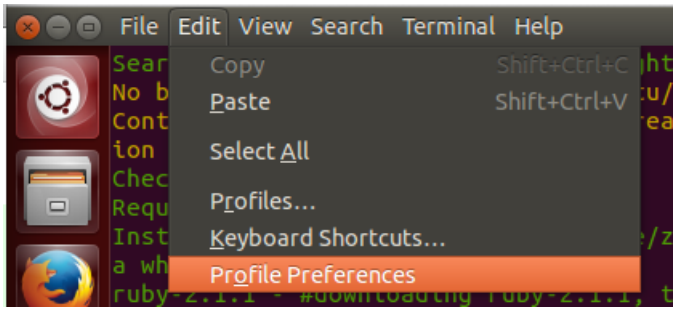
On Mac and Linux you can install and use multiple versions of Ruby at the same time using RVM[91] (Ruby Version Manager).

RVM on Ubuntu

On Ubuntu, first you have to install `curl`:

```
$ sudo apt-get install curl
...
```

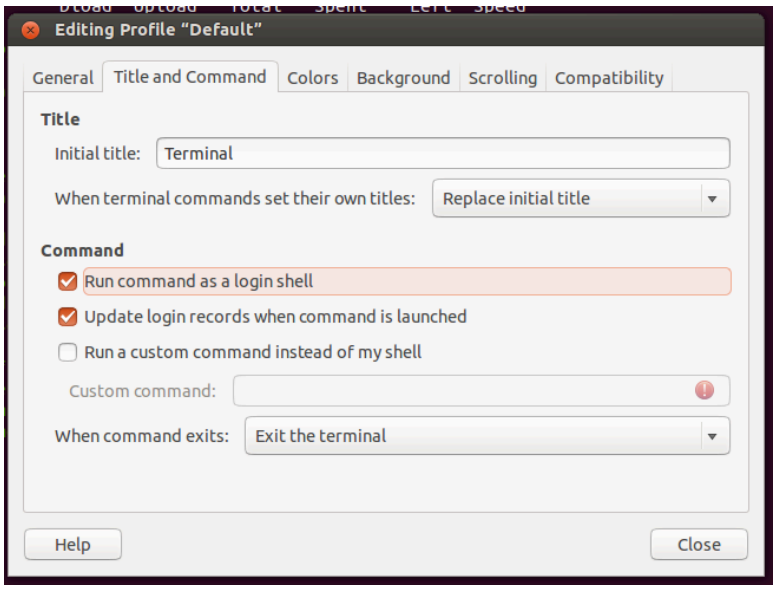
Then, you have to configure *Terminal*. Go to *Terminal* > *Edit* > *Profile Preferences*.



PhantomJS

[86] http://ruby-doc.org/stdlib-2.1.1/libdoc/pp/rdoc/PP.html
[87] https://www.ruby-toolbox.com/categories/irb_Alternatives
[88] https://www.ruby-toolbox.com/
[89] https://rubygems.org/gems/pry
[90] https://rubygems.org/gems/ripl
[91] http://rvm.io

Select *Title and Command* tab, check *Run command as a login shell* checkbox and then click *Close* button.



PhantomJS

In case of trouble see RVM documentation on Terminal[a].

[a] https://rvm.io/integration/gnome-terminal

Install RVM:

```
$ \curl -sSL https://get.rvm.io | bash -s stable
...
```

Open new *Terminal* tab or window. That is important. RVM might not work properly if you do not open new tab/window after installation. Finally, install a recent version of Ruby:

```
$ rvm install 2.1.1
...
```

Ask RVM which versions for Ruby are installed:

```
$ rvm list

rvm rubies

   ruby-2.1.0 [ i686 ]
   ruby-2.1.1 [ i686 ]
```

Terminal will not be aware of the Ruby installed via RVM until you explicitly tell it to use it. Try asking the *Terminal* for Ruby version:

```
$ ruby -v
The program 'ruby' can be found in the following
packages:
* ruby1.8
* ruby1.9.1
```

See, *Terminal* does not think Ruby is installed. Now, tell RVM you want to use the latest version:

```
$ rvm use ruby-2.1.1
Using /home/z/.rvm/gems/ruby-2.1.1
```

Ask *Terminal* again for Ruby version:

```
$ ruby -v
ruby 2.1.1p76 (2014-02-24 revision 45161)
[i686-linux]
```

Now it knows about Ruby installed via RVM. You can also ask RVM which Ruby is it using currently:

```
$ rvm list

rvm rubies

   ruby-2.1.0 [ i686 ]
=> ruby-2.1.1 [ i686 ]

# Default ruby not set. Try 'rvm alias create
default <ruby>'.

# => - current
# =* - current && default
#  * - default
```

Currently used Ruby (ruby-2.1.1) is marked with the arrow (=>).

The Ruby Toolbox

The Ruby Toolbox[92] is a really useful site where you can find and compare a lot of libraries. Libraries are grouped in categories. Whatewer you are looking for, it is probably

[92] https://www.ruby-toolbox.com

listed there. Some of categories and tools are: IRB alternatives[93] (like Pry[94]), PDF processing[95] (like PDF::Reader[96]), image processing[97] (like Chunky PNG[98]), log analysis[99] (like logstash[100]), dependency management[101] (like Bundler[102]), acceptance test frameworks[103] (like Cucumber[104]), browser testing[105] (like Selenium[106]), distributed testing[107] (like ParallelTests[108]) or unit test frameworks[109] (like RSpec[110]).

[93] https://www.ruby-toolbox.com/categories/irb_Alternatives
[94] http://pryrepl.org/
[95] https://www.ruby-toolbox.com/categories/PDF_Processing
[96] http://rubygems.org/gems/pdf-reader
[97] https://www.ruby-toolbox.com/categories/image_processing
[98] https://github.com/wvanbergen/chunky_png
[99] https://www.ruby-toolbox.com/categories/Log_Analysis
[100] http://logstash.net/
[101] https://www.ruby-toolbox.com/categories/dependency_management
[102] http://bundler.io/
[103] https://www.ruby-toolbox.com/categories/Acceptance_Test_Frameworks
[104] http://cukes.info/
[105] https://www.ruby-toolbox.com/categories/browser_testing
[106] https://rubygems.org/gems/selenium-webdriver
[107] https://www.ruby-toolbox.com/categories/distributed_testing
[108] https://github.com/grosser/parallel_tests
[109] https://www.ruby-toolbox.com/categories/testing_frameworks
[110] http://rspec.info/

Browser Developer Tools

TODO

Every contemporary browser has developer tools built in. This chapter will give a short overview of available options and how to use them.

Continuous Integration

TODO

There is only limited value in running tests here and there on your local machine. The best way to run tests is as frequently as possible using a continuous integration tool like Jenkins[111] or Travis CI[112]. In this chapter you will learn how to set up hosted Jenkins at CloudBees[113] and Travis CI.

[111] http://jenkins-ci.org/

[112] https://travis-ci.org/

[113] http://www.cloudbees.com/

Virtual Machines

TODO

It is really convenient to run tests in virtual machines. In this chapter you will learn how to use VirtualBox[114], Parallels[115], VMware[116] and Windows Virtual PC[117].

[114] https://www.virtualbox.org/
[115] http://www.parallels.com/
[116] http://www.vmware.com/
[117] http://www.microsoft.com/windows/virtual-pc/

Tools

TODO

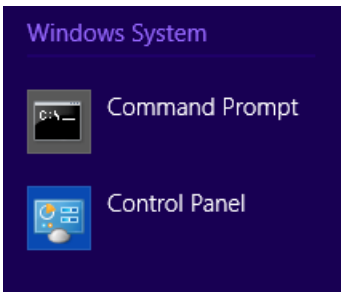
You will have to learn how to use Command-line interface[118] (CLI) since you will be spending a lot of time there. You will be spending a lot of time typing text so picking up a good text editor like Sublime Text[119] or IDE (integrated development environment) like RubyMine[120] will help. You will also be changing the code a lot so learning how to use a version control[121] tool like Git[122] is important. A lot of tools that are mentioned in this book are hosted at GitHub[123], Git hosting service.

Command-line interface

Command-line interface (CLI) is just another application. It is already installed, whether you are using Windows, Mac or Linux. CLI looks similar and behaves similarly on Mac and Linux, and it it slightly different on Windows.

Windows

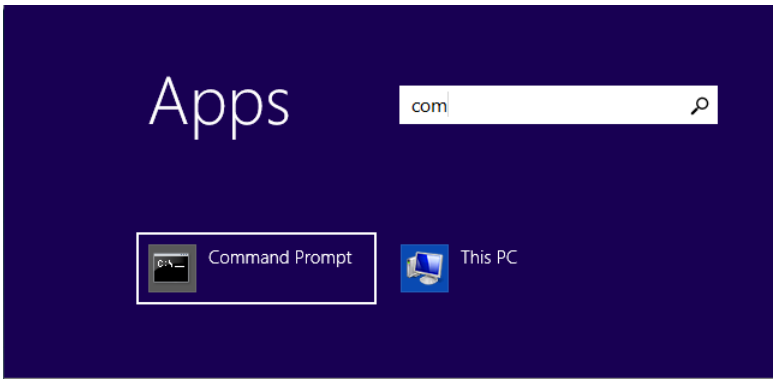
On Windows, the application is called *Command Prompt*. To open it (on Windows 8.1), go to *Apps* screen (swipe up, or click *down* arrow at the bottom of the screen) and it should be located in *Windows System* section.



***Command Prompt* app on Windows 8.1 *Apps* screen**

You can also search for it.

[118] https://en.wikipedia.org/wiki/Command-line_interface
[119] http://www.sublimetext.com/3
[120] http://www.jetbrains.com/ruby
[121] https://en.wikipedia.org/wiki/Revision_control
[122] http://git-scm.com
[123] https://github.com

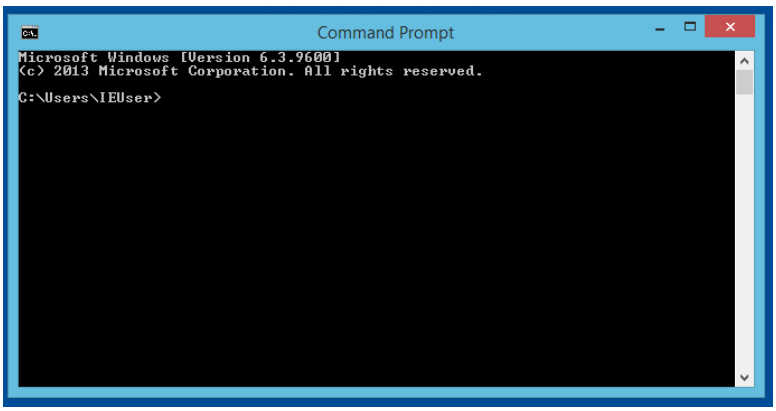


Searching for *Command Prompt* app on Windows 8.1

I could not find a keyboard shortcut to open *Command Prompt*.

By default, when you open *Command Prompt*, text similar to this will be displayed:

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights
reserved.

C:\Users\IEUser>
```



***Command Prompt* app on Windows 8.1**

You can ignore the first two lines. The last line, `C:\Users\IEUser>` is important. It is called *command prompt* (or just *prompt*). `C:\Users\IEUser` part is the folder where Command Prompt commands will be executed, also called *path*. `IEUser` is the name of the current user. The last character, `>`, separates path from the text that you will enter. Since both Mac and Linux use `$` as separator, this book will use that character. In short, if an instruction in the book says you should run a Ruby file with this command:

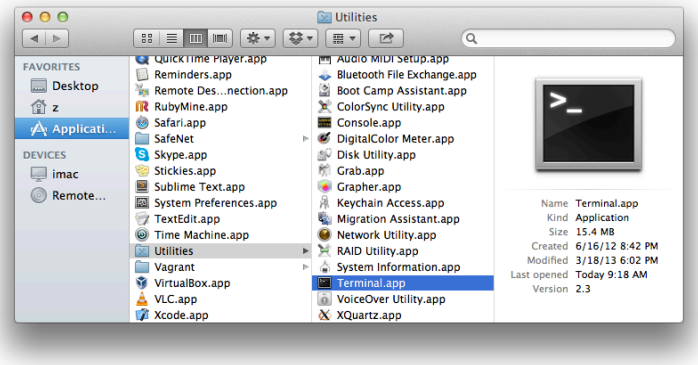
```
$ ruby file_name.rb
```

Your screen should look similar to this:

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights
reserved.

C:\Users\IEUser>ruby file_name.rb
```

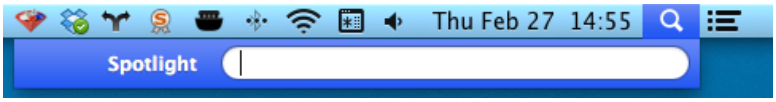
Mac

On Mac, CLI application is called *Terminal.app* and you can find it in `/Applications/Utilities` folder.



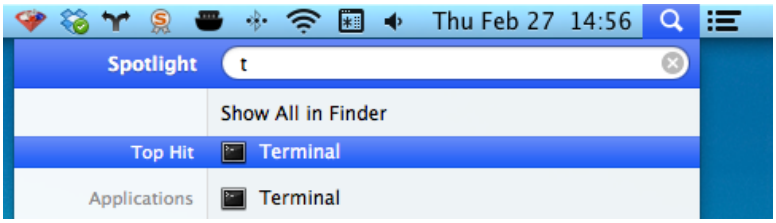
`Terminal.app` in *Finder*

A quick way to open it is by using built in *Spotlight* search. Press *cmd + space* or click magnifying icon and *Spotlight* will open.



Spotlight

Start typing `terminal` and you should see *Terminal* in search results.

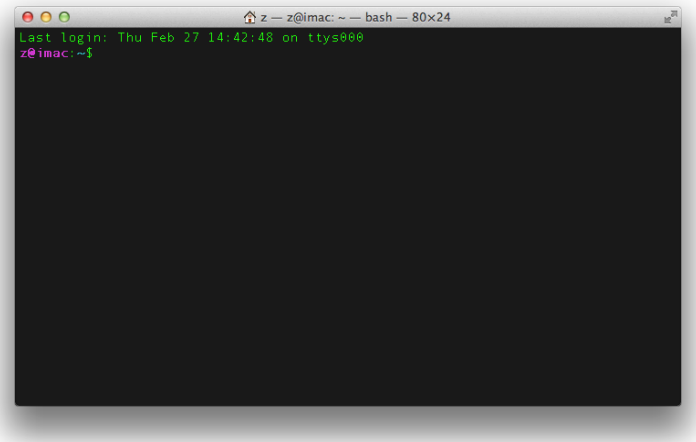


Terminal* app in *Spotlight

I could not find a keyboard shortcut to open *Terminal*.

When you open *Terminal*, text similar to this will be displayed:

```
Last login: Thu Feb 27 16:11:16 on ttys002
imac:~ z$
```



***Terminal* app on Mac**

You can ignore the first line. The second line contains several things. `imac` is machine name, ~ is the folder where *Terminal* commands will be executed, also called *path*. ~ is shortcut for `/Users/z`. `z` is the name of the current user. The last character, `$`, separates path from the text that you will enter. If an instruction in the book says you should run a Ruby file with this command:

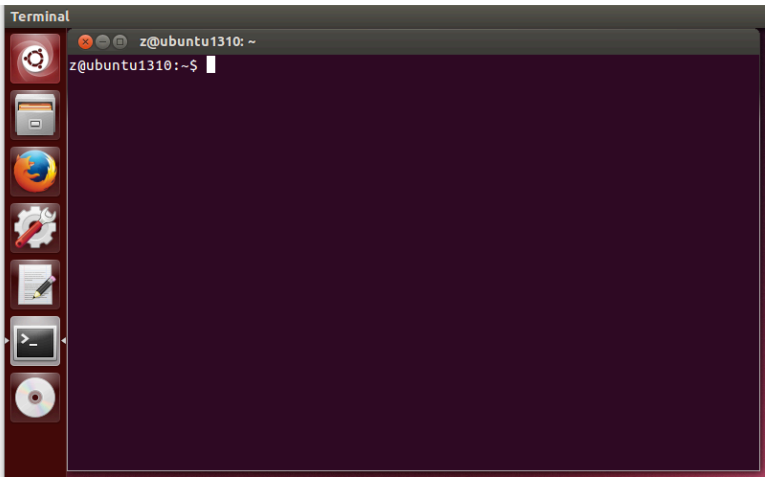
```
$ ruby file_name.rb
```

Your screen should look similar to this:

```
Last login: Thu Feb 27 16:11:16 on ttys002
imac:~ z$ ruby file_name.rb
```

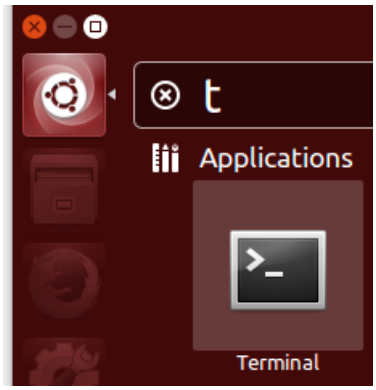
Ubuntu Linux

On Ubuntu Linux, the application is called *GNOME Terminal*. You can open it with keyboard shortcut *Ctrl + Alt + T*.



***Terminal* app on Ubuntu Linux**

Alternatively, search for *terminal* and you should find it.



***Terminal* app in Ubuntu search**

When you open *Terminal*, text similar to this will be displayed:

```
1 z@ubuntu1310:~$
```

Text contains several things. `z` is the name of the current user, `ubuntu1310` is machine name, ~ is the folder where *Terminal* commands will be executed, also called *path*. It is shortcut for `/home/z`. (Again, `z` is user name of the current user.) The last character, `$`, separates path from the text that you will enter. If an instruction in the book says you should run a Ruby file with this command:

```
1 $ ruby file_name.rb
```

Your screen should look similar to this:

```
1 z@ubuntu1310:~$ ruby file_name.rb
```

Contributors

- Alastair Montgomery: pull/3[124]
- Alex Rodionov[125]
- Andy Fong[126]
- Davor Banović[127]
- Dino Kovač[128]
- Felipe Knorr Kuhn: pull/1[129], pull/2[130]
- Kevin Emery[131]
- Kim Andi[132]
- Željko Filipin[133]

---

[124] https://github.com/watir/watirbook/pull/3
[125] https://github.com/watir/watirbook/commits?author=p0deje
[126] https://github.com/watir/watirbook/commits?author=afong
[127] https://github.com/watir/watirbook/commits?author=banovotz
[128] https://github.com/watir/watirbook/commits?author=reisub
[129] https://github.com/watir/watirbook/pull/1
[130] https://github.com/watir/watirbook/pull/2
[131] https://github.com/watir/watirbook/commits?author=kemery
[132] https://github.com/watir/watirbook/commits?author=msandi
[133] https://github.com/watir/watirbook/commits?author=zeljkofilipin

Changes

2014

0.9.2

- 2014-03-08
  - Added chapters: Driver, Element, Getting Help, Headless, RVM, IRB, Ruby Toolbox, CLI
  - Running scripts from the book at Travis CI

0.9.1

- 2014-01-25
  - Updated contributors, stats and todo.
  - Deleted unused files.
  - Added Quick Start chapter.

0.9.0

- 2014-01-18
  - Published PDF, EPUB, MOBI and HTML at Leanpub.
  - Renamed the book from "Homebrewer's Guide to Watir" to "Web Application Testing in Ruby (With Selenium and friends.)"
  - Installation chapters updated. Everything else removed from the book until updated.

2013

- 2013-11-14
  - Moved HTML version to https://leanpub.com/watirbook/read
  - Changed license to BSD.

2012

- 2012-11-28 Moved book repository to https://github.com/watir/watirbook and HTML version to http://watir.github.com/watirbook/
- 2012-11-22 The book finally has a usable HTML version at http://zeljkofilipin.github.com/watirbook
- 2012-11-21 Sold the first book via Leanpub.

0.8.0

- 2012-11-20 Published PDF, EPUB and MOBI at Leanpub, The first time the book is published at Leanpub.
  - New landing page at https://leanpub.com/watirbook
  - New or improved chapters: About, Link, Button, Image, Checkbox…
  - Created table with all supported HTML elements.
  - A lot of small improvements and fixes.
  - Renamed the book from “Watir Book” to “Homebrewer’s Guide to Watir”.
- 2012-11-17 Started working on the book again.
- 2012-11-14 Sold the last book via PayPal.

2011

0.7.1

- 2011-09-22 Published PDF, EPUB and MOBI at GitHub. Looks like this is the first time I have published free and paid versions of the book. Until now the entire book was available for free.

0.7

- 2011-09-17 Published PDF, EPUB and MOBI at GitHub.

0.6

- 2011-08-01 Published PDF, EPUB and MOBI at GitHub. Fixed images at GitHub.

0.5

- 2011-07-23 Published PDF, EPUB and MOBI at GitHub.

0.4.2

- ????-??-?? Started counting downloads.

0.4.1

- 2011-05-28 Published PDF, EPUB and MOBI at GitHub. The first time the book is published as MOBI.

0.4

- 2011-05-25 Published PDF and EPUB at GitHub.

0.3

- 2011-05-25 Published PDF and EPUB at GitHub.

0.2

- 2011-05-15 Published PDF and EPUB at GitHub.

0.3 installation

- 2011-03-23 Published PDF and EPUB at GitHub.

0.2 installation

- 2011-03-23 Published PDF and EPUB at GitHub. This version contains installation chapters only. The first time the book is published as EPUB.

0.1.3

- 2011-03-15 Published PDF at GitHub. Created book landing page at http://watir.com/book

0.1.2

- 2011-03-15 Published PDF at GitHub.

0.1.1

- 2011-03-15 Published PDF at Dropbox and GitHub.

0.1

- 2011-03-14 Published the book in PDF format at Dropbox. The first time the book is published as PDF and at at Dropbox. Sold the first book via PayPal.
- 2011-03-11 Started working on the book again.

2010

- 2010-04-27 Moved the book to GitHub. The first time the book is published as HTML and at GitHub.

2009

- 2009-11-16 First commit to the Git repository.
- 2009-11-08 Decided to write the book.

Stats

Updated 2014-01-25. Time is in hours, money in USD.

- royalties: 2,797.18
  - PayPal: 1,273.21
  - Leanpub: 1,523.97 == Money that book readers have donated to the Watir project
- hours worked: 215 (1.)
- USD/hour: 13 (2.)
- books sold: 377
  - PayPal: 144
  - Leanpub: 233
- books downloaded for free: 18,580
  - GitHub: 12,983 (3.)
  - Leanpub: 5,597
- records
  - maximum amount payed for one book: 100.00 (2012-03-06)
  - minimum amount payed for one book: 0.99 (8 times)
  - number of books sold in one day: 4 (2012-04-30, 2013-06-04)

1. probably double, but this is what I have in my notes
2. probably half, see (1.)
3. a bit more, I started counting downloads from version 0.4.2

License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name Željko Filipin nor the names of any other contributors to this software may be used to endorse or promote products derived from this software without specific prior written permission.

(based on BSD Open Source License)