Technical Background of Interactive CLI of Ruby 2.7

ITOYANAGI Sakura

RubyConf 2019

Powered by Rabbit 3.0.0 and COZMIXNG

Greeting

Hello, everyone.

Let me introduce myself

I'm

- a Ruby committer
- the current RDoc maintainer
- a member of Ruby core team

Community: Asakusa.rb

Asakusa.rb every Ruby Tuesday

Company: Space Pirates, LLC.

Space Pirates, LLC.

Company: Space Pirates, LLC.

Our business: We steal money via bank from venture companies that commission software development to us.

Company: Space Pirates, LLC.

This company is founded by my friend 2 years ago. Only 5 employees.

Company: Space Pirates, LLC.

...But it supported me as a semi-full time OSS engineer as a Ruby committer.

Hobby: Climbing

And my hobby is climbing.

Hobby: Climbing

Usually, I go to climbing area before international conference.

Hobby: Climbing

But this time, I couldn't go to climbing before RubyConf.

Hobby: Climbing

Because I went to Matsue where Matz is living to attend the RubyWorld Conference as a speaker.

Hobby: Climbing

And I told about "adventure".

Hobby: Climbing

Adventure is to go somewhere that nobody hasn't known the world.

Hobby: Climbing

Nobody understands the value, nobody knows how can we go there.

Hobby: Climbing

And everyone is living in **well-known** comfort zones, but adventure is not.

Hobby: Climbing

Only one week later after the presentation of the RubyWorld Conference, I came here. So I couldn't climb around Nashville.

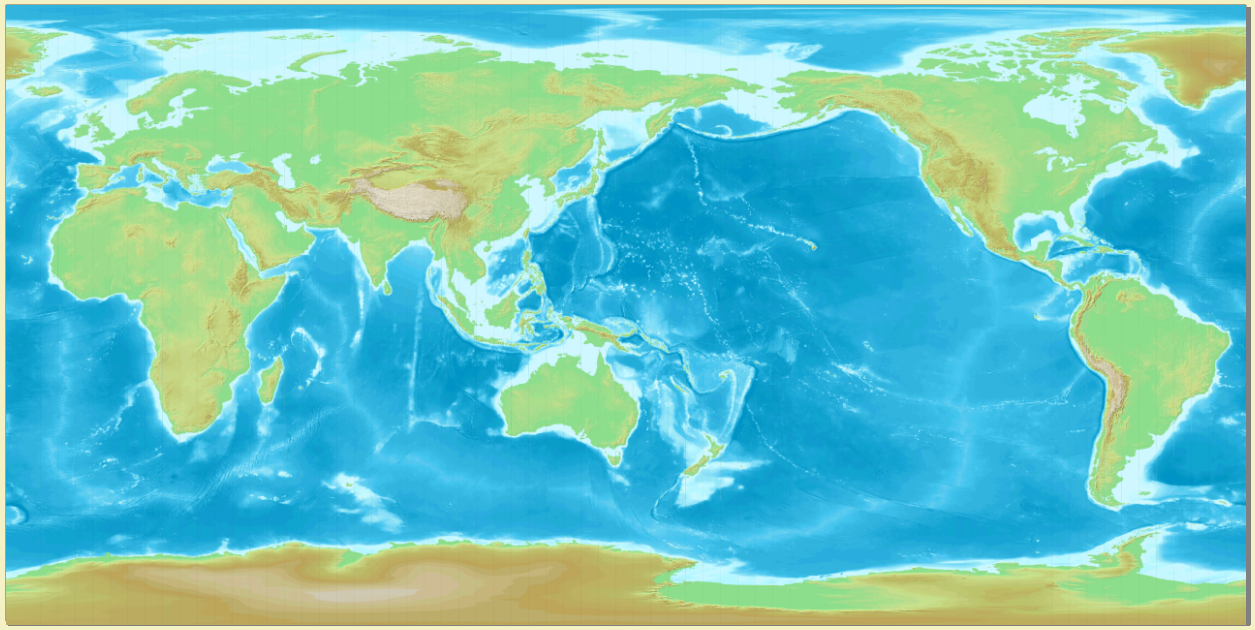
Hobby: Climbing

But I found a good place to climb near here.

Hobby: Climbing

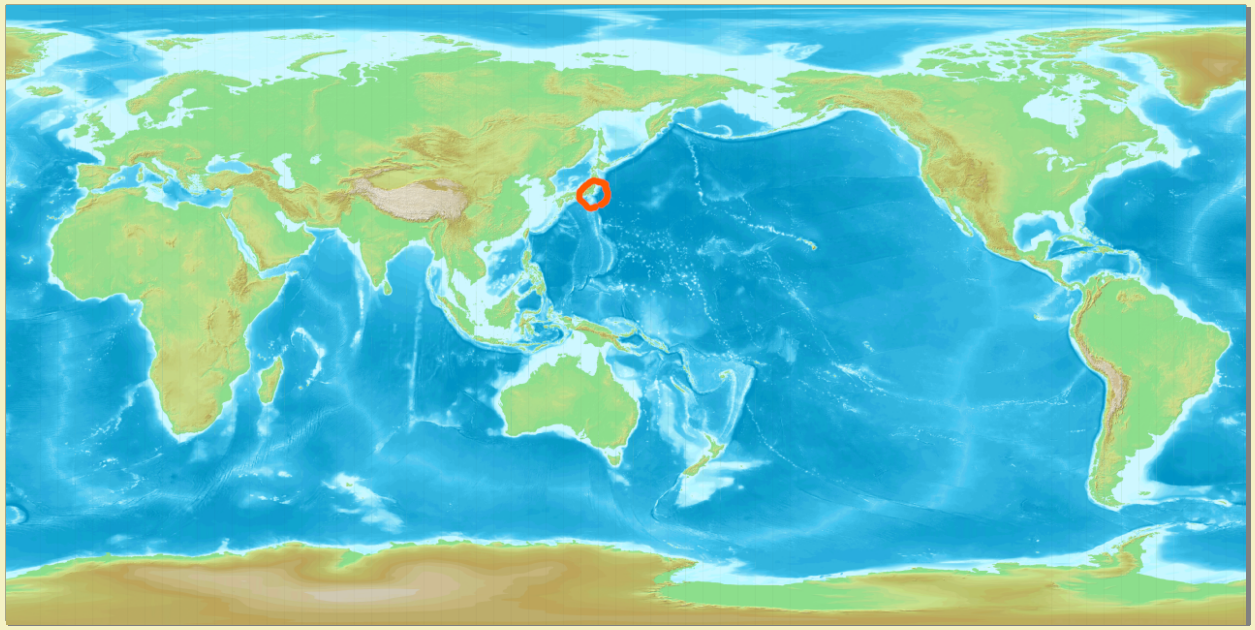
It's Puerto Rico.

Hobby: Climbing



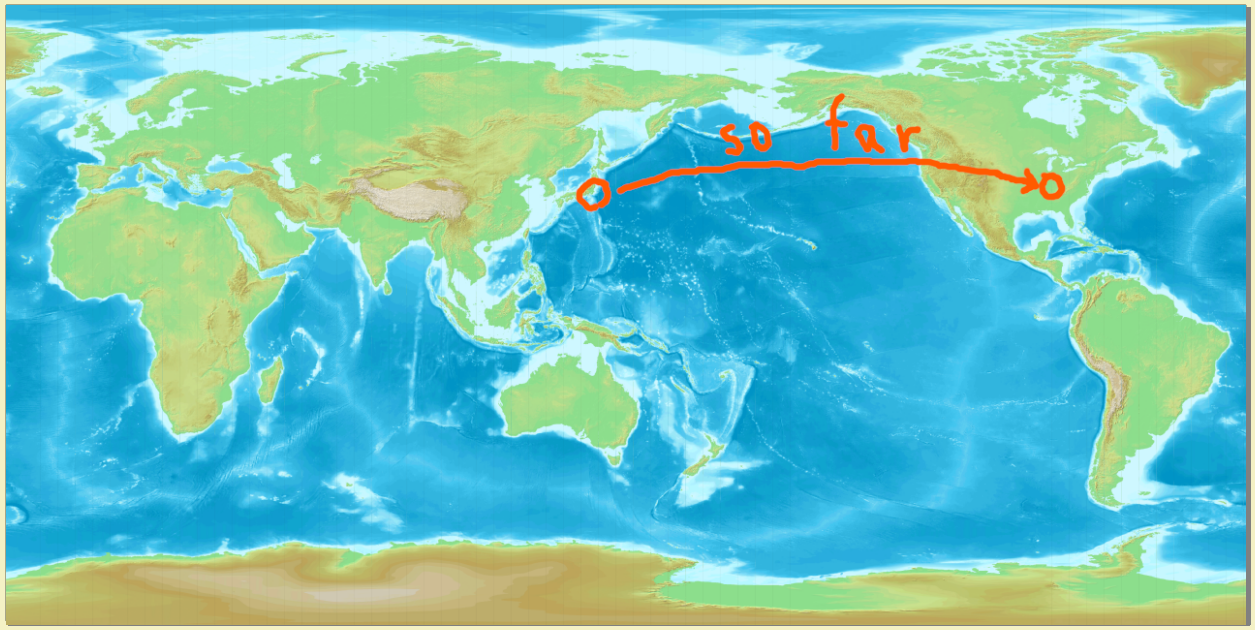
world map

Hobby: Climbing



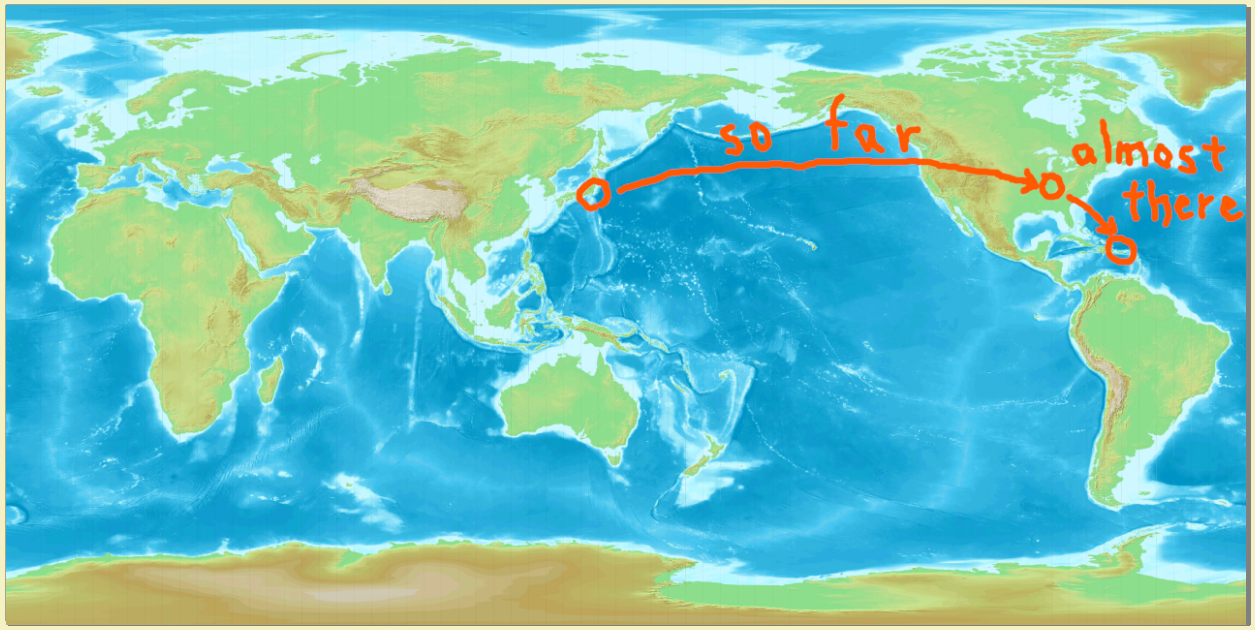
I'm from Japan.

Hobby: Climbing



And it's Nashville. So far.

Hobby: Climbing



Puerto Rico is almost there.

Hobby: Climbing

I'll try to climb **unknown** and unexplored area of a jungle of Puerto Rico.

Hobby: Climbing

The word, unknown is important for adventure.

Hobby: Climbing

I think that adventure means going into the unknown.

My Adventure In Ruby

Today, I'll talk about my adventure in Ruby.

My Adventure In Ruby

I'm the current maintainer of RDoc which is the standard documentation tool of Ruby.

My Adventure In Ruby

And I'm trying to improve IRB with documentation.

My Adventure In Ruby

The brand-new IRB has multi-line editings that is powered by Reline.

My Adventure In Ruby

The multi-line editing feature of IRB was advocated by keiju-san who is the author of the original IRB.

My Adventure In Ruby

It's the great vision but it's too hard to implement because the original IRB is implemented by GNU Readline.

My Adventure In Ruby

GNU Readline has over 30 years of histrical background.

My Adventure In Ruby

So Reline needs to be compatible with so many features of GNU Readline.

My Adventure In Ruby

- the history of terminal
- GNU Readline compatible features
- I18n support

My Adventure In Ruby

- **the history of terminal**
- GNU Readline compatible features
- I18n support

The History of Terminal

- the history of terminal
  - the Morse code
  - typewriter
  - teletype
  - escape sequence
  - escape sequence on Unix like OS
  - Windows support

The History of Terminal

When do you think the terminal's historical background started?

- 30 years ago?
- 60 years ago?
- 120 years ago?
- 240 years ago?

The History of Terminal

Most communication technologies are invented by market of new businesses.

The History of Terminal

Japanese people continues to eat rice over 10,000 years. It's our soul. Old Japanese kings treat rice stockpiles as assets.

The History of Terminal

Back then, rice is a practical currency in Japan.

The History of Terminal

About 200 years ago, merchant of those days was in trouble.

The History of Terminal

Rice market has different between east side and west side.

The History of Terminal

So they needed the soonest communication technology.

The History of Terminal

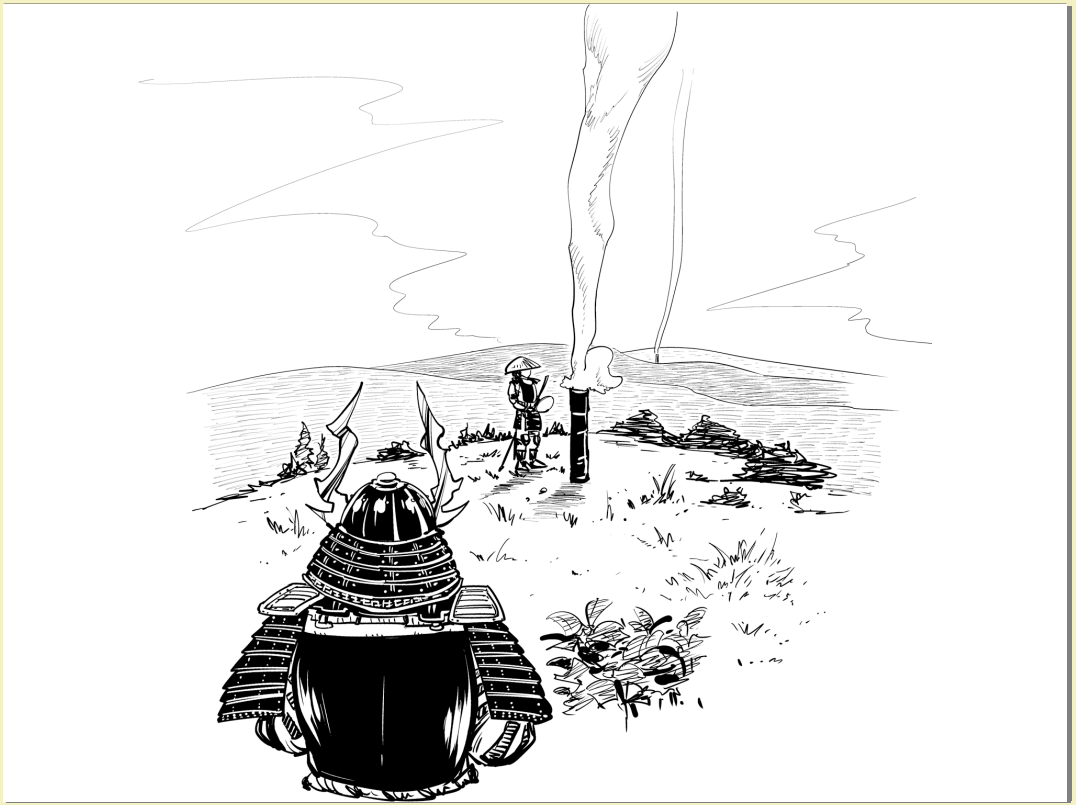
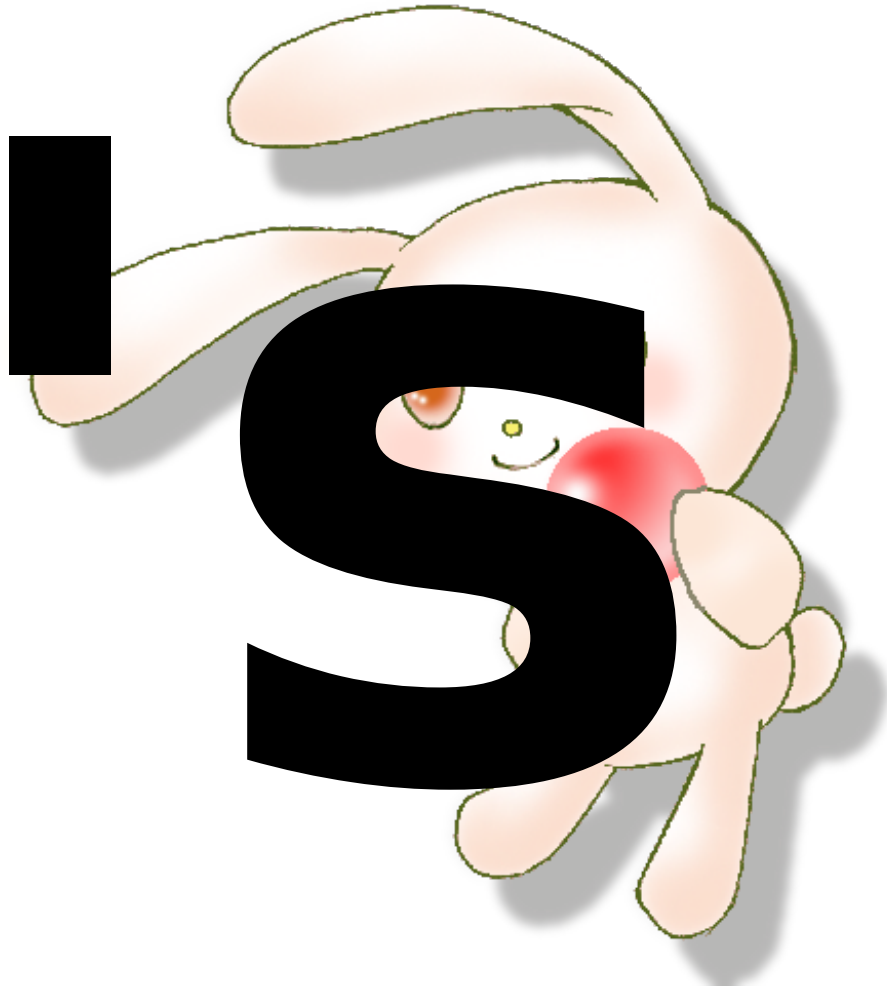
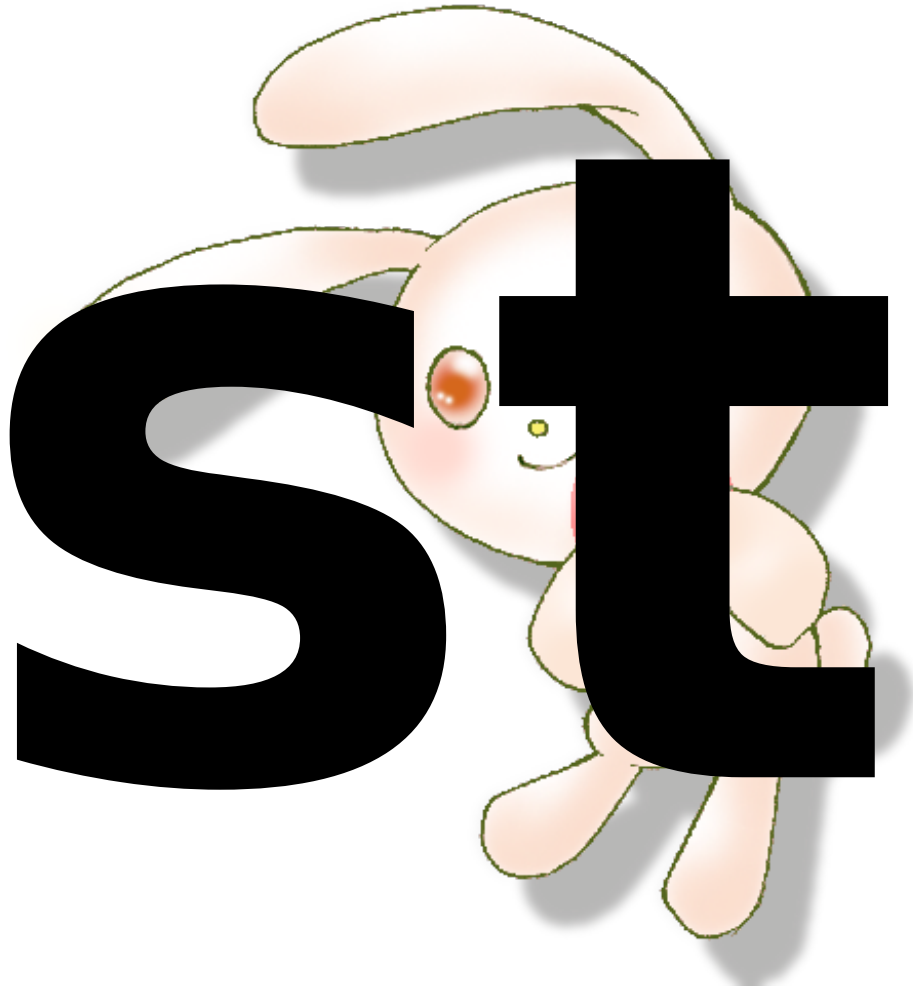


Illustration purpose by © 2019 Doom Kobayashi

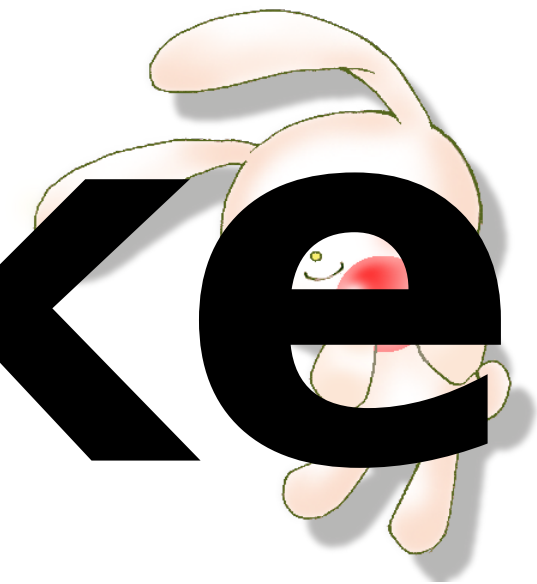
It's

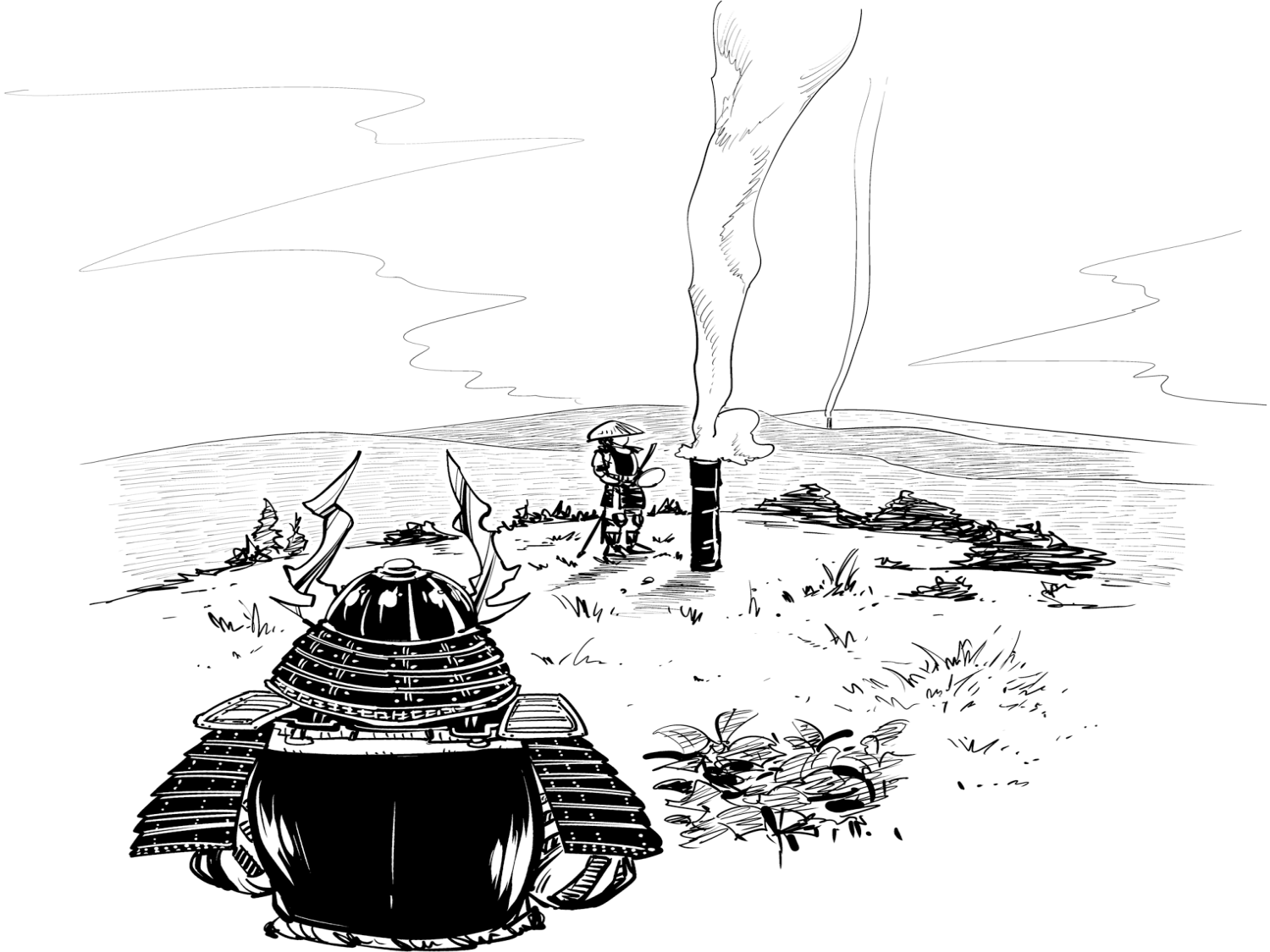


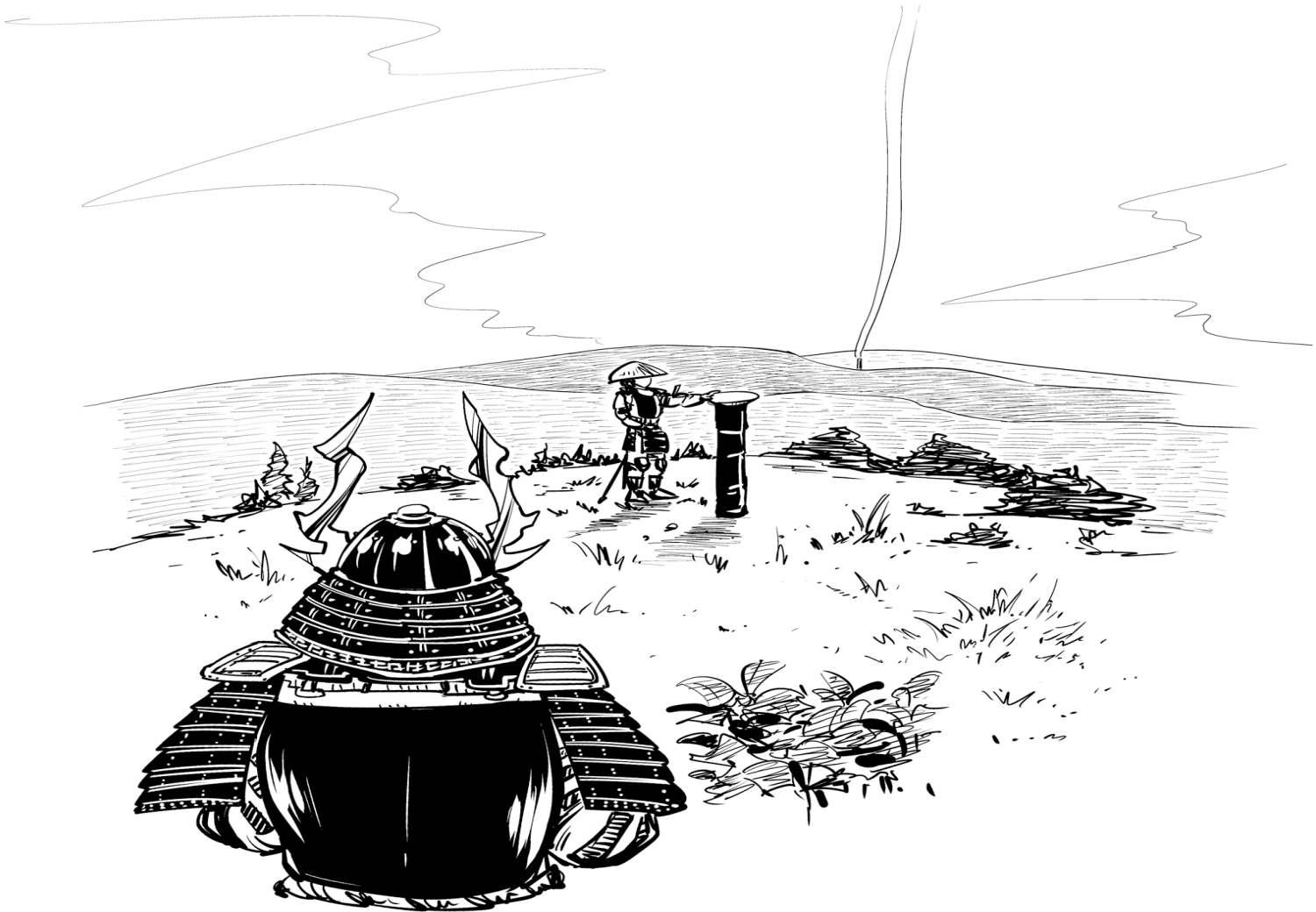
just



smoke







The History of Terminal

It's a kind of bit encoded data.

The History of Terminal

Merchants could send rice market information within 2 hours over 500km.

The History of Terminal

In the same age, telegraph is invented by William F. Cooke and Charles Wheatstone.

The History of Terminal

It sends code from typed primitive keys via railway track as a line to a printing system.

The History of Terminal

Cooke and Wheatstone's five-needle, six-wire telegraph

The History of Terminal

It's just experimental so it has only several keys. It's not enough to type alphabet, so "shift key" is added.

The History of Terminal

It's the "shift key" in early times. It was 1837.

The History of Terminal

After that, Samuel Morse who is famous Morse code invents telegraph on Morse code.

The History of Terminal

The system is just Morse code so can receive generated code from a typed key or hand inputted code, and can output to auto printing system or writing characters via ear.

The History of Terminal

The system continues to be improved, it's called "teletype".

The History of Terminal

Royal Earl House invented brand new teletype and it's used for money transfer. It was 1855. A few years later, The Western Union Company is founded.

The History of Terminal

But the typing system and printing system is not convenient.

The History of Terminal

Human beings know more convenient typing and printing system.

The History of Terminal

It's...

The History of Terminal

Typewriter

The History of Terminal

But typewriter needs "operations of a roll paper".

The History of Terminal

Typewriters print characters to the same point but move a roll paper. The protocol that ups to here doesn't contain operations of a roll paper.

The History of Terminal

- Move left
- Move right
- Roll a paper(move to next line)
- Move to head of line
- ...

The History of Terminal

Those operations are added to the protocol.

The History of Terminal

- Move left
- Move right
- Roll a paper(move to next line)
- Move to head of line

The History of Terminal

- Move cursor left
- Move cursor right
- Line feed
- Carriage return

The History of Terminal

These are "control codes".

The History of Terminal

The reason of those two operations are separated is those need too many time to finish.

- Line feed
- Carriage return

The History of Terminal

Aside, "Line break" character code is...

- Carriage return + Line feed on Windows
- Carriage return on macOS
- Line feed on Unix like OSes

The History of Terminal

The difference is based on early times operations set of printing systems for each OSes.

The History of Terminal

Now, other some operations are added to the protocol. It's the base of modern "terminal". It was 1901.

The History of Terminal

The early "terminal" was that separated "keyboard" and "printing system" from typewriter.

The History of Terminal

The "printing system" is the base of "line printer".

The History of Terminal

And, some terminals need "extended features". So, a new character, "following characters are not printable, just control code" is added to the protocol.

The History of Terminal

These are called "escape key" and "escape sequence".

The History of Terminal

But many companies develop new "terminal" machines. They specify non-compatible escape sequences each other.

The History of Terminal

It's a flood of terminals. Users are confused hardly.

The History of Terminal

In those times, a new technology comes.

The History of Terminal

It's...

computer

The History of Terminal

Teletype terminals and line printers come to be connected to computers, eventually, line printers are replaced with visual monitors.

The History of Terminal

"Desk Set"(1957), sponsored by IBM

The History of Terminal

Many escape sequences for terminals are different so computers support them by hardware because softwares is still immature.

The History of Terminal

Dozens of years later, primitive softwares come to be OSes. Unix comes up. User space on OS changes "settings" of software.

The History of Terminal

Unix like OSes changed the situation of escape sequences.

The History of Terminal

Termcap what is encapsulated software for incompatible escape sequences named each escape sequence, and has a dictionary from name to real escape sequence.

The History of Terminal

It's a revolution. Users can use any terminals for own computer. It's developed at 1978.

The History of Terminal

And Terminfo what is improved Termcap is developed at 1982.

The History of Terminal

> *ANSI sequences were introduced in the 1970s to replace vendor-specific sequences and became widespread in the computer equipment market by the early 1980s.*
>
> *[cited from `ANSI escape code - Wikipedia']*

The History of Terminal

Especially, SGR parameters is famous to set character decoration.

The History of Terminal

```
print "\e[31m" # red
print "red"
print "\e[32m" # green
print "green"
print "\e[34m" # blue
print "blue"
print "\e[0m" # reset
print "\n"
```

result:

redgreenblue

The History of Terminal

This is the very sad history of terminals, but Windows introduced another way.

The History of Terminal

Windows has Console API for control terminal as known as command prompt.

The History of Terminal

Console API of Windows controls a console via "console handle".

The History of Terminal

Escape sequences need using I/O to control console.

The History of Terminal

Console API of Windows is smarter API for console, it's very practical!

The History of Terminal

And it means Console API is a newcomer of the terminal's sad history.

The History of Terminal

It's complex insanely.

The History of Terminal

Humans are stupid.

The History of Terminal

I asked a question at the start of this section.

"When do you think the terminal's historical background started?"

The History of Terminal

An answer is "unclear".

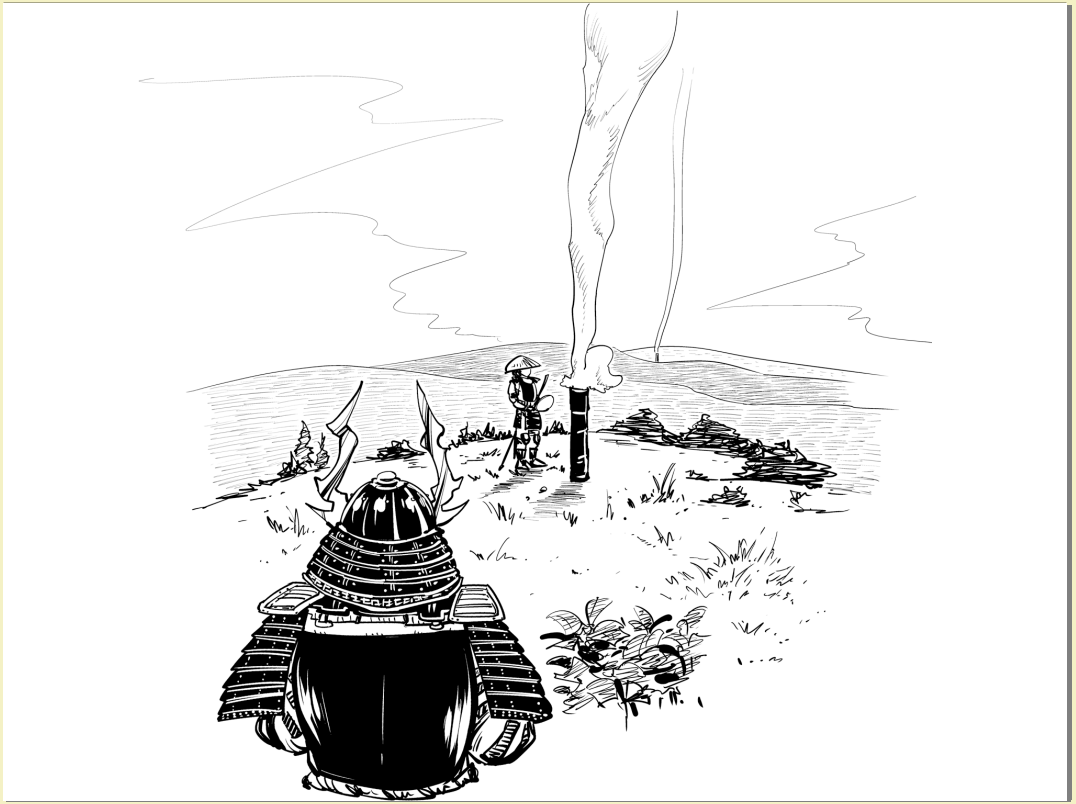
The History of Terminal

What is "terminal"?

What is "the protocol"?

What is "encoded data"?

The History of Terminal



The History of Terminal

Maybe, fire's smoke is the earliest long distance communication technology.

My Adventure In Ruby

- the history of terminal
- GNU Readline compatible features
- I18n support

My Adventure In Ruby

- the history of terminal
- **GNU Readline compatible features**
- I18n support

GNU Readline Compatible Features

Ruby needs GNU Readline as a native library.

GNU Readline Compatible Features

GNU Readline is powerful line editor for taking user input.

GNU Readline Compatible Features

```
require 'readline'



Readline.readline('prompt>')
```

Shows the prompt and reads the inputted

GNU Readline Compatible Features

Line editing is...:

- Move cursor
- Delete characters
- Use history
- ...

GNU Readline Compatible Features

```ruby
# small IRB sample
require 'readline'

while (line = Readline.readline('echo>'))
  break if line == 'exit'
  print eval(line) # evaluate!
end
```

GNU Readline Compatible Features

GNU Readline is used by...:

- shell(tcsh, Bash, and others)
- MySQL command-line tool
- The GNU Debugger(GDB)

GNU Readline Compatible Features

Ruby's standard library "readline" is used by...:

- IRB
- Pry
- Thor(famous simple framework for command line utilities)

GNU Readline Compatible Features

The "readline" library is very important for Ruby. But "readline" can be used only when GNU Readline is installed before Ruby builds.

GNU Readline Compatible Features

```
# Ubuntu/GNU Linux case
$ sudo apt install libreadline-dev
$ rbenv install 2.6.5
```

If you forget installing "libreadline-dev" first, Ruby doesn't have "readline" library.

GNU Readline Compatible Features

```
$ pry # tried to launch Pry without readline lib
Sorry, you can't use Pry without Readline or a compatible library.
Possible solutions:
 * Rebuild Ruby with Readline support using `--with-readline`
 * Use the rb-readline gem, which is a pure-Ruby port of Readline
 * Use the pry-coolline gem, a pure-ruby alternative to Readline
```

Pry fails to launch when Ruby doesn't have "readline" library.

GNU Readline Compatible Features

It's must be a trap to beginners. So I decided to re-implement "readline" library by pure Ruby. It's Reline.

GNU Readline Compatible Features

Ruby 2.7 uses GNU Readline by default, and uses Reline inside if doesn't have GNU Readline.

GNU Readline Compatible Features

Reline has 3 layers:

- Keyboard input
- Line editing
- Build string as default encoding of the environment

GNU Readline Compatible Features

- **Keyboard input**
- Line editing
- Build string by default encoding of the environment

Reline uses select(2) system call in Unix like OSes, kbhit() and getwch() in Windows Console API, to take keyboard input.

GNU Readline Compatible Features

- Keyboard input
- **Line editing**
- Build string by default encoding of the environment

And I ported Emacs bindings and Vi bindings from GNU Readline for line editing.

GNU Readline Compatible Features

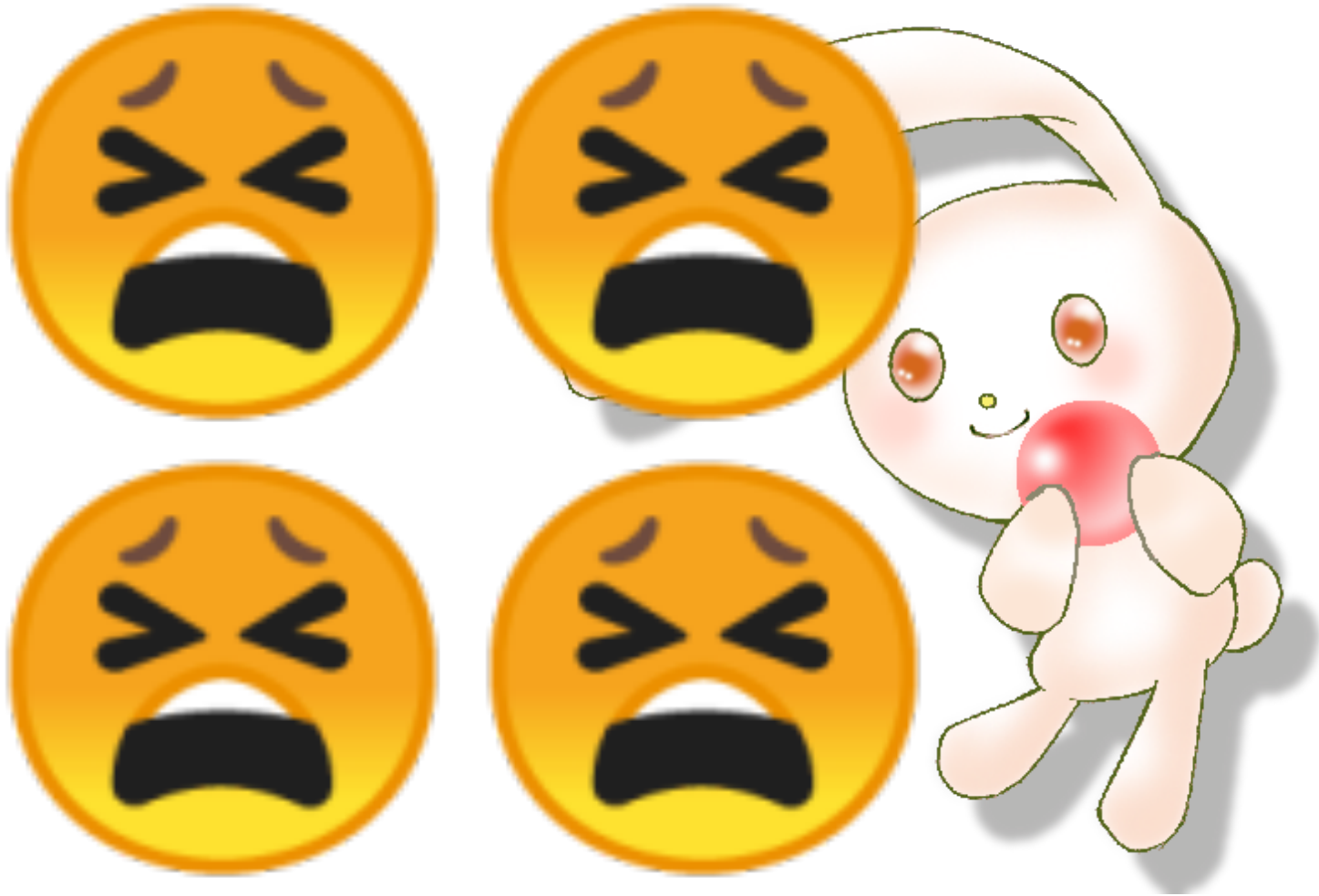
- Keyboard input
- Line editing
- **Build string by default encoding of the environment**

Finally, I implemented building string as the default encoding of the environment.

GNU Readline Compatible Features

- Keyboard input
- Line editing
- **Build string by default encoding of the environment**

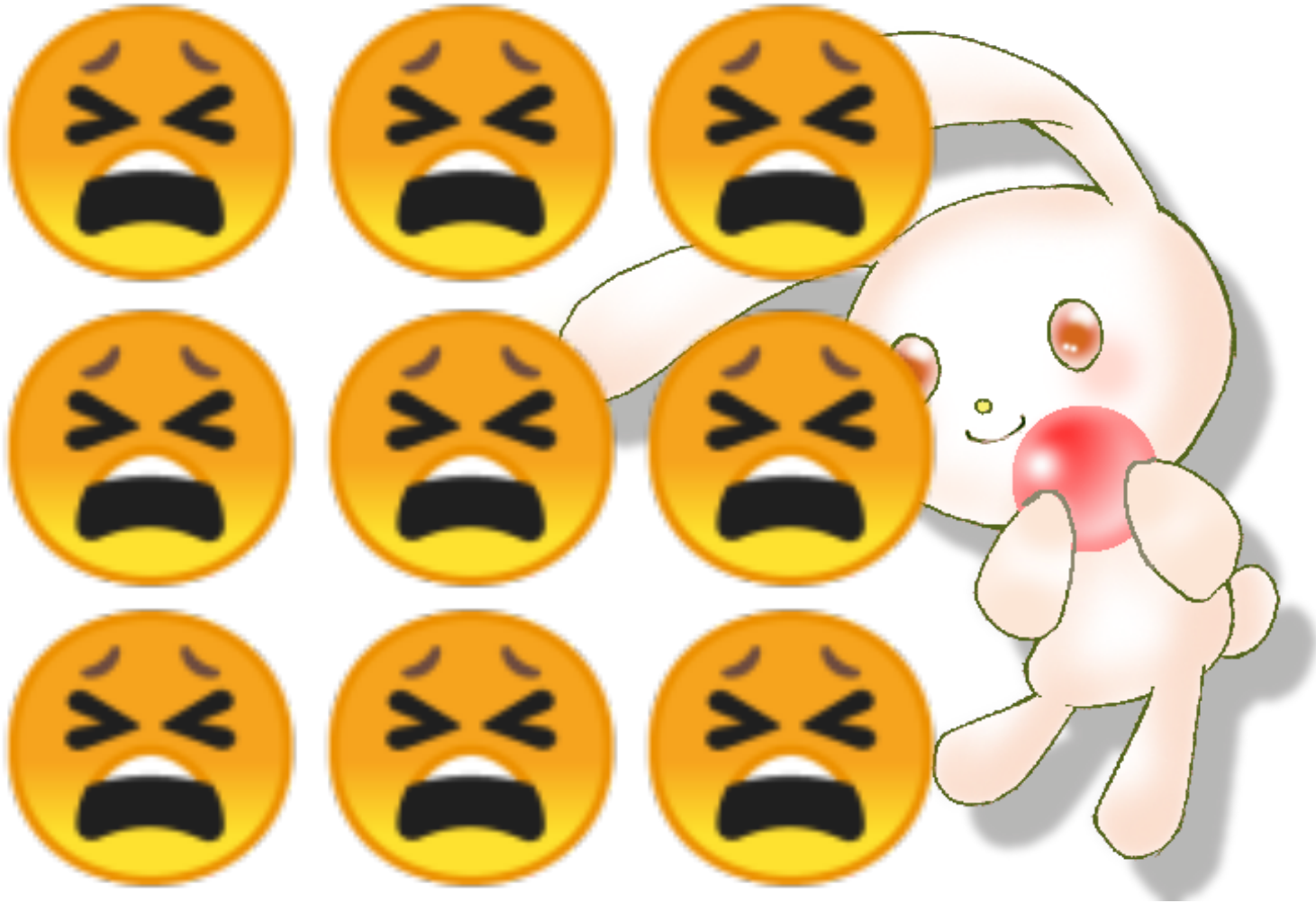
I got off from work! I did it!



GNU Readline Compatible Features

- Keyboard input
- **Line editing**
- Build string by default encoding of the environment

But the implementation is broken in non-Unicode encodings, so I re-implement whole line editting code.



GNU Readline Compatible Features

- **Keyboard input**
- Line editing
- Build string by default encoding of the environment

Unicode characters are broken at the time of first input...I fixed it...

GNU Readline Compatible Features

- Keyboard input
- **Line editing**
- Build string by default encoding of the environment

Combining Unicode charasters are sometimes broken in line editing...

GNU Readline Compatible Features

- Keyboard input
- Line editing
- **Build string by default encoding of the environment**

I fixed the whole implementation the layer due to lower layer...

GNU Readline Compatible Features

- **Keyboard input**
- **Line editing**
- **Build string by default encoding of the environment**

All tests fail so I remake whole tests.

GNU Readline Compatible Features

- **Keyboard input**
- **Line editing**
- **Build string by default encoding of the environment**

I worked out over 2 years but I'm still fixing source code and tests.

GNU Readline Compatible Features

I consult Ruby core team about the implementation problems, and almost finished.

GNU Readline Compatible Features

It will be adopted at Ruby 2.7.

GNU Readline Compatible Features

But there is still some work to be done.

GNU Readline Compatible Features

It's Reidline.

GNU Readline Compatible Features

The original author of IRB, keiju-san, he's developing new IRB, it's Reirb.

GNU Readline Compatible Features

Reirb uses an original line editor "Reidline" inside.

GNU Readline Compatible Features

Reidline is a **multiline** editor, like JavaScript console in browser.

GNU Readline Compatible Features

But the implementation is too hard, so I added Reidline mode to Reline. It's just for Reirb but Ruby 2.7's IRB contains the Reidline mode as a transition period.

My Adventure In Ruby

- the history of terminal
- GNU Readline compatible features
- **I18n support**

I18n Support

There are so many character encoding in the world, especially CJK(Chinese, Japanese, Korean) have so complex characters and history. More than 10,000 Kanji characters, Kana, Hangul...

I18n Support

But it's very confused for non CJK people.
So I'll try explain by emoji's specifications.

I18n Support

We always use the word "character" primitively. But it's a very difficult thing.

I18n Support

It's important to understand the difference between codepoint and grapheme in Unicode but it confuses you.

I18n Support

Some codepoints are invisible because these are just "combining character" for "base character".

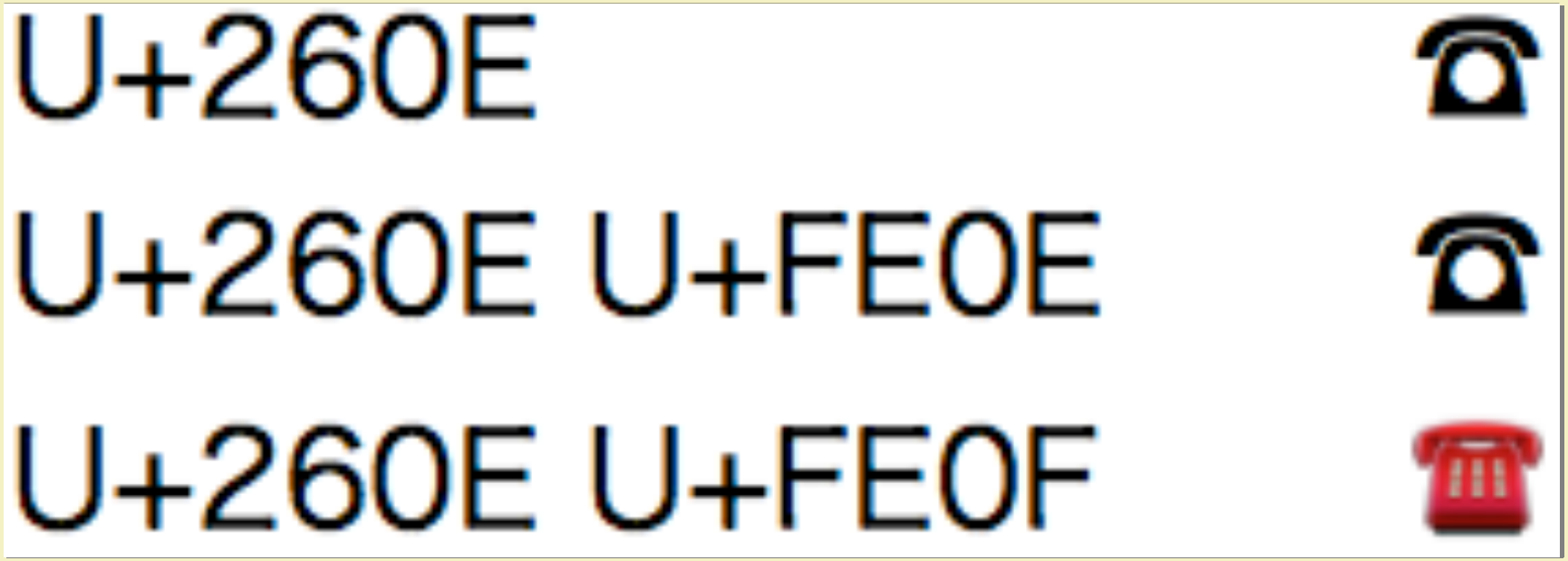
I18n Support

For example, "☎"(U+260E BLACK TELEPHONE) is changed with following invisible "variation selector" if you use a font that has the "variation".

I18n Support

For example, the "variation" is "textual fashion"(U+FE0E VARIATION SELECTOR-15) or "emoji fashion"((U+FE0F VARIATION SELECTOR-16)).

I18n Support



I18n Support

And some combining characters has a glue codepoint(U+200D ZERO WIDTH JOINER) to join different characters.

I18n Support

For example, "  "(EYE IN SPEECH BUBBLE U+1F441 U+FE0F U+200D U+1F5E8 U+FE0F) is composed of "eye"(U+1F441 EYE) and "  "(U+1F5E8 LEFT SPEECH BUBBLE) with a glue codepoint(U+200D ZERO WIDTH JOINER).

I18n Support

```
$ irb
irb(main):001:0> eye = "\u{1F441}"
=> "👁"
irb(main):002:0> left_speech_bubble = "\u{1F5E8}"
=> "🗨"
irb(main):003:0> emoji_fashion = "\u{FE0F}"
=> ""
irb(main):004:0> eye + emoji_fashion
=> "👁"
irb(main):005:0> left_speech_bubble + emoji_fashion
=> "🗨"
irb(main):006:0> glue = "\u{200D}"
=> ""
irb(main):007:0> eye + emoji_fashion + glue + left_speech_bubble + emoji_fashion
=> "👁‍🗨"
```

I18n Support

Besides, national flags are constructed by alphabets.

I18n Support

"  "(U+1F1FA U+1F1F8 flag for United States) is composed of "🇺"(U+1F1FA REGIONAL INDICATOR SYMBOL LETTER U) and "🇸"(U+1F1F8 REGIONAL INDICATOR SYMBOL LETTER S) **without joiner**.

I18n Support

DEMO

I18n Support

Unicode has contains human's confused history.

I18n Support

So, the "codepoint" is an unit that should be coded.

I18n Support

And the "grapheme" is an unit that human beings understand as a character.

I18n Support

-  - 2 codepoints, 1 grapheme
- **U** - 1 codepoint, 1 grapheme
- **S** - 1 codepoint, 1 grapheme
- US(ASCII) - 2 codepoints, 2 graphemes
- U+200D(ZWJ) - 1 codepoint, 0 grapheme
-  - 5 codepoints, 1 grapheme

I18n Support

String#chars method returns codepoints.

String#grapheme_clusters method returns graphemes.

```
"🇺🇸".chars                # => ["U", "S"]
"🇺🇸".grapheme_clusters    # => ["🇺🇸"]
```

I18n Support

Do you understand?

I18n Support

I have no confidence.

I18n Support

If Reline remove only 1 codepoint from 1 grapheme that is constructed by plural codepoints, the editor break easily.

My Adventure In Ruby

...It's an outline of technical background of interactive CLI of Ruby.

My Adventure In Ruby

The brand-new IRB will be adopted at Ruby 2.7.

My Adventure In Ruby

And, I'll release the brand-new IRB before Ruby 2.7.

My Adventure In Ruby

```
$ gem install irb
$ irb # brand-new IRB!
```

After that, you can install and use the brand-new IRB.

My Adventure In Ruby

When will I release the brand-new IRB?

Right Now

My Adventure In Ruby

```
$ gem install irb
```

Install the brand-new IRB.

DEMO of the brand-new IRB

My Adventure In Ruby

```
$ gem install irb
```

Install the brand-new IRB.

Right Now.

My Adventure In Ruby

Please file some issues if you find bugs.

- https://github.com/ruby/irb
- https://github.com/ruby/reline

Take it easy. It's a great contribution for us.