# Ruby をビルドする方法

田中 哲
@tanaka_akr

# 326 種類の Ruby をビルドする方法 ~0.49 から 2.6.0-preview1 まで ~

田中 哲
@tanaka_akr

# 327 種類の Ruby をビルドする方法 ~0.49 から 2.6.0-preview2 まで ~

田中 哲
@tanaka_akr

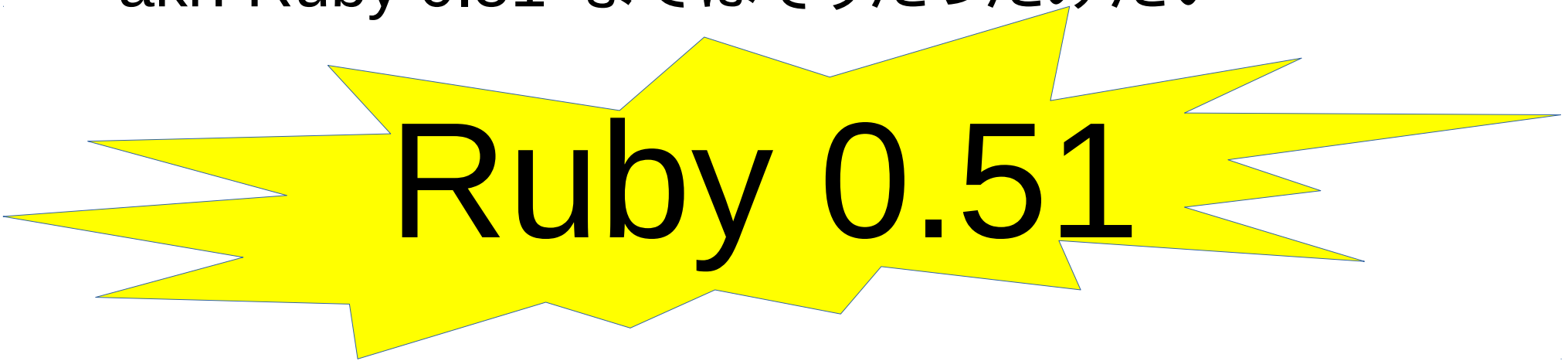
# all-ruby

https://github.com/akr/all-ruby

# 古い Ruby をビルドすると どのように嬉しいか

Ruby 開発者会議における例 :
[Feature#14724] chains of inequalities
a < b && b < c を a < b < c と書けるようにしようという提案についての議論

- usa: 単に a < b が b を返せばいいんじゃね？
- matz: 実は昔の Ruby はそうだったんだけど
- akr: Ruby 0.51 まではそうだったみたい



# all-ruby の使い方

```
% all-ruby -e 'print(1 < 2)'
ruby-0.49               2
...
ruby-0.51               2
ruby-0.54               t
...
ruby-0.95               t
ruby-0.99.4-961224    TRUE
...
ruby-1.1a7             TRUE
ruby-1.1a8             true
...
ruby-2.6.0-preview1   true
```

たしかに １＜２ が
２ を返している！

# all-ruby の使い方

```
% all-ruby -e 'print(1 < 2)'
ruby-0.49               2
...
ruby-0.51               2
ruby-0.54               t
...
ruby-0.95               t
ruby-0.99.4-961224    TRUE
...
ruby-1.1a7             TRUE
ruby-1.1a8             true
...
ruby-2.6.0-preview1    true
```

Lisp っぽい
まぁ Ruby は Lisp だしね

# all-ruby の使い方

```
% all-ruby -e 'print(1 < 2)'
ruby-0.49               2
...
ruby-0.51               2
ruby-0.54               t
...
ruby-0.95               t
ruby-0.99.4-961224    TRUE
...
ruby-1.1a7             TRUE
ruby-1.1a8             true
...
ruby-2.6.0-preview1   true
```

昔は大文字で
表示されてたのか

# all-ruby の使い方

```
% all-ruby -e 'print(1 < 2)'
ruby-0.49               2
...
ruby-0.51               2
ruby-0.54               t
...
ruby-0.95               t
ruby-0.99.4-961224    TRUE
...
ruby-1.1a7             TRUE
ruby-1.1a8             true
...
ruby-2.6.0-preview1   true
```

いつも見慣れた
面白みのない挙動

# print を使っているのは 昔は p がなかったから

```
% all-ruby -e 'p 1 < 2'
ruby-0.49           -e:1: syntax error
            #<Process::Status: pid 21654 exit 1>
ruby-0.50           -e:1: syntax error
            #<Process::Status: pid 21655 exit 1>
ruby-0.51           -e:1: undefined method `p' for "main"(Object)
            #<Process::Status: pid 21656 exit 1>
ruby-0.54           -e:1:in method `p': undefined method `p' for "main"(Object)
            #<Process::Status: pid 21657 exit 1>
ruby-0.55           -e:1: undefined method `p' for "main"(Object)
            #<Process::Status: pid 21658 exit 1>
...
ruby-0.76           -e:1: undefined method `p' for "main"(Object)
            #<Process::Status: pid 21665 exit 1>
ruby-0.95           -e:1: undefined method `p' for main(Object)
            #<Process::Status: pid 21666 exit 1>
ruby-0.99.4-961224    TRUE
...
ruby-1.1a7           TRUE
ruby-1.1a8           true
...
ruby-2.6.0-preview1   true
```

メソッド呼び出しに括弧が必須なので syntax error

p がないので undefined method

微妙にエラーメッセージも変わっている

# p に感謝せよ

# TRUE って使えるの？

```
% all-ruby -e 'print(TRUE)'
ruby-0.49               nil
...
ruby-0.65               nil
ruby-0.69               t
...
ruby-0.95               t
ruby-0.99.4-961224    TRUE
...
ruby-1.1a7             TRUE
ruby-1.1a8             true
...
ruby-2.4.0-preview1   true
ruby-2.4.0-preview2   -e:1: warning: constant ::TRUE is deprecated
                      true
...
ruby-2.6.0-preview1   -e:1: warning: constant ::TRUE is deprecated
                      true
```

エラーになってないけど
nil は変だろう

警告

# もしかして昔は未定義定数の参照はエラーじゃなかった？

```
% all-ruby -e 'print(XXX)'
ruby-0.49               nil
...
ruby-0.65               nil
ruby-0.69               -e:1: Uninitialized constant XXX
                  #<Process::Status: pid 8574 exit 1>
...
```

やはり未定義定数が nil
perl っぽい

# Ruby をビルドする方法

- git clone https://github.com/akr/all-ruby.git
- cd all-ruby
- rake all

# 実際は 32bit 開発環境が必要

- 32bit 開発環境の設定 (Debian GNU/Linux の場合)
  - sudo dpkg --add-architecture i386
  - sudo apt update
  - sudo apt-get build-dep ruby2.3
  - sudo apt install rake gcc-multilib \
    zlib1g:i386 libncurses5:i386 libgdbm3:i386 libssl1.0.2:i386 libreadline7:i386 libffi6:i386
- git clone https://github.com/akr/all-ruby.git
- cd all-ruby
- rake all

# rake all に時間がかかることは御承知置きください

- rake all はすべてのバージョンの ruby を
  - ダウンロードし
  - ビルドします
  - ( テストはしません )
- でも rake なので、一回やってしまえば、次回からは差分だけで済みます
- 我慢できなければ、@shyouhei & @hsbt 作の docker image を御利用ください
- あと @mametter が slack で bot を動かしている模様です

# 古い Ruby を素直にビルドできる？

- もちろんできない
- Ruby 1.6 までは 32bit 開発環境が必要
- それ以外にもいろいろ調整が必要
  Rakefile と patch/ で調整している
  - OpenSSL の非互換
  - varargs は gcc 3.3 以降では使えない
  - いろいろ細かな修正

# OpenSSL の非互換

- OpenSSL は非互換に変化する
- 新しい Ruby は追随してコンパイルできるようにする（がんばれ Ruby 開発者）
- 古い Ruby は古いままなのでコンパイルできない
- 対処 : OS (Debian) に入っている OpenSSL 1.1.0 を使えない Ruby では ext/openssl を disable する
- OpenSSL 1.1.0 を使えるのは Ruby 2.4.0 からなので、それ以前の Ruby の ext/openssl はあきらめる
- いずれまた OpenSSL が変わったら、あきらめるバージョンをあげればいい

# varargs

- varargs は K&R C で可変引数関数を書く方法
- ANSI C で stdarg が定義された
- varargs は gcc 3.3 以降では使えない
- ruby-1.1b9_18 までは varargs が必須
- ruby-1.1b9_19 からは stdarg を使えるが、
  古い Ruby は古いまま
- 書き換えはたいして難しくないが、パッチだと周辺の変化であたらなくなり、バージョンごとにパッチを作るのは面倒
- 対処 : varargs から stdarg に乱暴に書き換えるコードを書いた
  （古い ruby に対してだけ動けばいいのでなんとかなる）

# convert_varargs_to_stdarg ( 抜粋 )

- src.gsub!(/^#include <varargs.h>\n/, <<-End.gsub(/^\s*/, ''))
  #ifdef __STDC__
  #include <stdarg.h>
  #define va_init_list(a,b) va_start(a,b)
  #else
  #include <varargs.h>
  #define va_init_list(a,b) va_start(a)
  #endif
  End
- src.gsub!(/va_start\(args\);/, 'va_start(args, fmt);')
- 簡単な部分だけ抜粋した
- 簡単でない部分もある

# 細かな修正

- cont-elif.diff
- defines-nodbm.diff
- error-error.diff
- error-error2.diff
- extmk-heredoc.diff
- glob-alloca.diff
- glob-alloca2.diff
- gnuglob-alloca.diff
- gnuglob-alloca2.diff
- gnuglob-dirent.diff
- instruby-dll.diff
- makefile-assoc.diff
- math-define-erange.diff
- parse-semicolon.diff
- rb_type-definition.diff
- rbconfig-expand.diff
- regex-re_match_2.diff
- ruby-errno.diff
- ruby-errno2.diff
- ruby-errno3.diff
- signal-unistd.diff
- struct-va_end.diff
- struct-va_end2.diff
- tcltklib-extconf.diff
- time-time.diff
- time-time2.diff
- variable-break.diff

どのバージョンに対してどれをあてるか
ハードコードしてある
パッチの中身はどれもたいしたことはしていない

# 3 個ビルドをあきらめた

- ruby-0.62.tar.gz, ruby-0.63.tar.gz
  gzip 形式じゃないといわれて展開できない
- ruby-0.71.tar.gz
  hash.c がない
- 他はビルドできた

# 将来課題

- 古い OpenSSL を自前でビルドして使う？

# なぜ all-ruby を作ったのか？

all-ruby スクリプトと Rakefile は開発のきっかけが違う

- all-ruby (2009 ～ )
  - 昔から ruby のリリースごとにビルドを残していた
  - それらをいっきに実行するスクリプトを作った
- Rakefile (2014 ～ )
  - 手元の環境が 32bit(i386) から 64bit(amd64) に変化
  - 古い 64bit 未対応 ruby はビルド不能になった
  - 古い環境からバイナリをコピーすればとりあえず動くが、再現不能なのでビルドしなおす方法を調べた
  - 必要に応じて 32bit 開発環境でビルドする Rakefile を作った
  - いきおいあまって Ruby 0.49 までコンプリート (3 個無理だったから完全コンプリートではないけれど )

# 再現可能性は重要<br>あと、chkbuild との関連

- だれでも (Debian で ) 再現可能な Rakefile を公開
- @shyouhei, @hsbt, @mametter といったユーザが発生
- @shyouhei は以前、chkbuild でたくさんの Ruby ビルドを残していたこともあり、たくさんビルドというのは琴線に触れるものがあるのだろう
- @mametter もコードカバレッジに関して chkbuild をいじったことがある
- @hsbt は現在最大の chkbuild ユーザ
- chkbuild と all-ruby はどちらも大量に Ruby をビルドする
- フィットするところは違うにせよ、似たところはあるのかもしれない

# 関連研究

- 青木さんの bitclust に forall-ruby があったらしい
- usa さんによれば、すべての ruby committer は all-ruby っぽいスクリプトを持っているらしい
- multiruby というものもあったようだ
- chariesome/historical-rubies
  0.49, 1.0-971225, 1.2.6 のパッチのみ
- rhenium/ruby-ancient 0.49 のみ
- usa: ruby0.49 とか常備しておくのはたしなみ
- ドキュメントにメソッドが作られたバージョンなどを書こうとしたこともあったが、追随されず、ドキュメントから消されるなどであまりうまくいかなかった

# 閑話:考古学

- Unix version 7 の make を現在の環境でビルドしたことがある
  アーカイブ (ar) のサポートをあきらめたらなんとかなった
  - UnixArchive はすばらしい
- Ruby を VAX でビルドしたこともある
  - simh はすばらしい
  - NetBSD はすばらしい
  - ネットワークは重要
  - VAX 浮動小数点には NaN がない
- Ruby を s390 でビルドしたこともある
  - hercules はすばらしい
  - Debian はすばらしい
  - 16 進浮動小数点は体験できなかった
- あ、もちろん qemu もすばらしい

# まとめ

- rake はすばらしい
  ルールをプログラムで生成できるのがいい
- 非互換はよくない
- 考古学は楽しい
- コンプリートは射幸心を煽る
- @hsbt が示したように all-ruby は有用であるから、考古学には実用的な意味がある