🚀 Curso Completo

Migración de Ruby 2 a Ruby 3

Agosto 2025 - Versión 3.0
Actualizado para Ruby 3.3.8

Tabla de Contenidos

1. □ Introducción y Preparación

Bienvenido al Curso Completo de Migración Ruby 2 → Ruby 3

Este curso te guiará paso a paso en el proceso de migración de Ruby 2 a Ruby 3, cubriendo todos los aspectos técnicos, mejores prácticas y herramientas necesarias para una migración exitosa.

1.1 ¿Por qué migrar a Ruby 3?

□ **Beneficios Clave de Ruby 3:**

- **Rendimiento:** Hasta 3x más rápido que Ruby 2.0
- **Concurrencia:** Ractor para verdadero paralelismo
- **Typing:** RBS para tipado estático opcional
- **Compatibilidad:** 99% compatible con Ruby 2.7
- **Seguridad:** Mejoras en manejo de memoria

1.2 Versiones y Cronología

Versión	Fecha de Lanzamiento	Soporte hasta	Estado
Ruby 2.0	Febrero 2013	Febrero 2016	□ Sin soporte
Ruby 2.1	Diciembre 2013	Marzo 2017	□ Sin soporte
Ruby 2.2	Diciembre 2014	Marzo 2018	□ Sin soporte
Ruby 2.3	Diciembre 2015	Marzo 2019	□ Sin soporte
Ruby 2.4	Diciembre 2016	Marzo 2020	□ Sin soporte
Ruby 2.5	Diciembre 2017	Marzo 2021	□ Sin soporte

Ruby 2.6	Diciembre 2018	Marzo 2022	□ Sin soporte
Ruby 2.7	Diciembre 2019	Marzo 2023	□ Sin soporte
Ruby 3.0	Diciembre 2020	Marzo 2024	□ Sin soporte
Ruby 3.1	Diciembre 2021	Marzo 2025	□ Mantenimiento
Ruby 3.2	Diciembre 2022	Marzo 2026	□ Soporte completo
Ruby 3.3	Diciembre 2023	Marzo 2027	□ Soporte completo

1.3 Preparación del Proyecto

□ Lista de Verificación Pre-Migración:

- ☑ Backup completo del código fuente
- ☑ Documentar versiones actuales de gemas
- ☐ Crear rama específica para migración
- ☐ Configurar entorno de testing robusto
- ☐ Identificar dependencias críticas
- ☐ Planificar rollback strategy
- ☐ Comunicar timeline al equipo

□ Ejercicio 1: Evaluación Inicial

Objetivo: Evaluar el estado actual de tu proyecto Ruby.

```
# Ejecutar en terminal para obtener información del proyecto ruby --version bundle --version gem list --local git log --oneline -10 find . -name "*.rb" | wc -l grep -r "Ruby" Gemfile*
```

Resultado esperado: Un reporte detallado del estado actual de tu aplicación Ruby.

2. □ Historia y Evolución de Ruby 3

Ruby 3x3: La Visión Cumplida

Ruby 3 representa la culminación de años de desarrollo enfocado en el rendimiento, la concurrencia y la compatibilidad. La famosa promesa "Ruby 3x3" buscaba hacer Ruby 3 veces más rápido que Ruby 2.0.

2.1 Hitos Principales

Característica	Versión Introducida	Impacto	Estado en Ruby 3
MJIT (JIT Compiler)	Ruby 2.6	Rendimiento	□ Mejorado significativamente
Ractor	Ruby 3.0	Concurrencia	□ Paralelismo real
Fiber Scheduler	Ruby 3.0	Async/IO	□ I/O no bloqueante
RBS (Type Signatures)	Ruby 3.0	Tipos	□ Tipado estático opcional
TypeProf	Ruby 3.0	Análisis	□ Inferencia de tipos

2.2 Cambios Disruptivos (Breaking Changes)

⚠ **Cambios que Rompen Compatibilidad:**

- **Argumentos posicionales y por palabra clave separados**
- **Eliminación de constantes: Fixnum, Bignum, NIL, TRUE, FALSE**
- **Remoción de métodos taint/untaint**

- **Cambios en Proc y lambda**
- **Modificación en el comportamiento de $1, $2, etc.**

2.3 Benchmark de Rendimiento

```
# Ejemplo de mejora de rendimiento require 'benchmark' def fibonacci(n) return n if n <= 1 fibonacci(n-1) + fibonacci(n-2) end # Ruby 2.7 vs Ruby 3.3 - Fibonacci(35) # Ruby 2.7: ~4.2 segundos # Ruby 3.3: ~1.4 segundos (3x más rápido) Benchmark.bm do |x| x.report("Fibonacci(35):") { fibonacci(35) } end
```

Mejoras de Rendimiento Documentadas:

- **Operaciones aritméticas:** 2-3x más rápidas
- **Llamadas a métodos:** 1.5-2x más rápidas
- **Asignación de objetos:** 1.3x más eficiente
- **Garbage Collection:** Latencia reducida 40%
- **I/O operations:** Hasta 10x en casos async

3. □ Preparación del Entorno

Configuración Profesional para Migración

Una migración exitosa requiere un entorno bien preparado con las herramientas correctas y procesos establecidos.

3.1 Instalación de Ruby 3

Usando rbenv (Recomendado)

```
# Actualizar rbenv cd ~/.rbenv && git pull # Listar versiones disponibles rbenv install --list | grep 3\. # Instalar Ruby 3.3.8 (última estable) rbenv install 3.3.8 # Configurar como global o local rbenv global 3.3.8 # o para proyecto específico: rbenv local 3.3.8 # Verificar instalación ruby --version # ruby 3.3.8 (2024-04-09 revision 12345) [x86_64-linux]
```

Usando RVM

```
# Actualizar RVM rvm get stable # Instalar Ruby 3.3.8 rvm install ruby-3.3.8 # Usar la nueva versión rvm use ruby-3.3.8 --default # Crear gemset específico para migración rvm gemset create migration rvm use ruby-3.3.8@migration
```

Usando Docker (Para Testing)

```
# Dockerfile para testing FROM ruby:3.3.8-alpine WORKDIR /app COPY Gemfile* ./ RUN bundle install COPY . . # Comando para testing ENTRYPOINT ["bundle", "exec"]
```

3.2 Configuración de Bundle

```
# Gemfile actualizado source 'https://rubygems.org' ruby '3.3.8' # Especificar versión requerida # Gemas core que pueden necesitar actualización gem 'webrick', '~> 1.8' # Requerido para servidor web gem 'psych', '~> 5.0' # YAML parser gem 'fiddle', '~> 1.1' # FFI wrapper gem 'stringio', '~> 3.0' # String I/O operations # Resto de tu aplicación...
```

```
# Configurar bundle para desarrollo bundle config set --local path 'vendor/bundle' bundle config set --local jobs 4 bundle config set --local retry 3 # Instalar dependencias bundle install # Verificar compatibilidad bundle exec ruby -c Gemfile bundle doctor
```

3.3 Herramientas de Análisis

```
# Instalar herramientas de migración gem install ruby_migrator gem install rubocop-migration gem install bundler-audit # Herramientas específicas para Ruby 3 gem install rbs gem install typeprof gem install steep
```

3.4 Configuración de Testing

```
# spec/spec_helper.rb o test/test_helper.rb # Configurar warnings para detectar deprecations if RUBY_VERSION >= "3.0" Warning.process(__FILE__) do |warning| # Log warnings para análisis posterior File.open("migration_warnings.log", "a") do |f| f.puts "[#{Time.now}] #{warning}" end # Mostrar warnings en desarrollo warn warning if Rails.env.development? end end # Configurar variable de entorno para testing ENV['RUBY_VERSION'] = RUBY_VERSION ENV['MIGRATION_MODE'] = 'true'
```

🎯 Ejercicio 2: Configuración de Entorno

Objetivo: Configurar un entorno completo para migración.

```
#!/bin/bash # setup_migration_env.sh echo "🚀 Configurando entorno de migración Ruby 2→3" # 1. Verificar versiones actuales echo "Versiones actuales:" ruby --version bundle --version gem --version # 2. Crear backup del Gemfile cp Gemfile Gemfile.ruby2.backup cp Gemfile.lock Gemfile.lock.ruby2.backup # 3. Crear rama de migración git checkout -b ruby3-migration # 4. Configurar bundle para nueva versión bundle config set --local path 'vendor/bundle' bundle config set --local jobs $(nproc) echo "✅ Entorno preparado para migración"
```

Resultado esperado: Entorno completamente configurado con backups y herramientas instaladas.

4. Análisis de Compatibilidad

Identificación Sistemática de Problemas

Antes de realizar cambios, es crucial identificar todos los puntos de incompatibilidad en tu código base.

4.1 Análisis Automático con Ruby Migrator

```
# Instalar Ruby Migrator gem install ruby_migrator # Análisis básico (solo reporte) ruby_migrator --path mi_proyecto --report-only # Análisis detallado con formato JSON ruby_migrator --path mi_proyecto --format json --verbose > reporte.json # Análisis específico de directorio ruby_migrator --path app/models --report-only --verbose
```

Ejemplo de Reporte de Análisis

```
 Ruby 2→3 Migration Analysis Report ═══════════════════════════════════════ Summary: • Files analyzed: 45 • Errors found: 12 • Warnings: 18 • Info items: 6 • Total issues: 36  Critical Issues (Errors): ├── app/models/user.rb:15 - Removed constant 'Fixnum' ├── app/helpers/format.rb:23 - Deprecated method 'taint' ├── config/application.rb:8 - Positional/keyword argument issue └── lib/utils.rb:45 - Invalid lambda syntax ⚠ Warnings: ├── app/views/layouts/app.html.erb:12 - Hash syntax (upgrade recommended) ├── spec/models/user_spec.rb:34 - Global variable usage ($1, $2) └── Gemfile:23 - Gem version may be incompatible ℹ Informational: ├── README.md - Update Ruby version references └── .ruby-version - Update to 3.3.8
```

4.2 Análisis Manual con Grep

```
#!/bin/bash # manual_analysis.sh - Búsqueda manual de patrones problemáticos echo " Análisis manual de compatibilidad Ruby 3" # Buscar constantes removidas echo "▶ Constantes removidas:" grep -r "Fixnum\|Bignum\|NIL\|TRUE\|FALSE" app/ lib/ --include="*.rb" # Buscar métodos deprecados echo "▶ Métodos deprecados:" grep -r "\.taint\|\.untaint\|\.trust\|\.untrust" app/ lib/ --include="*.rb" # Buscar sintaxis de hash antigua echo "▶ Sintaxis hash antigua:" grep -r ":\w\+\s*=>" app/ lib/ --include="*.rb" # Buscar variables globales problemáticas echo "▶ Variables globales:" grep -r '\$[1-9][0-9]*' app/ lib/ --include="*.rb" # Buscar usage de lambda/proc potencialmente problemático echo "▶ Lambda/Proc usage:" grep -r "lambda\|proc\|Proc\.new" app/ lib/ --include="*.rb"
```

4.3 Análisis de Dependencias

```
# bundle_analyzer.rb - Análisis de gemas require 'bundler' puts "📦 Análisis de Gemas para Ruby 3" puts "═══════════════════════════════" Bundler.load.specs.each do |spec| puts "#{spec.name} (#{spec.version})" # Verificar soporte Ruby 3 if spec.required_ruby_version supports_ruby3 = spec.required_ruby_version.satisfied_by?(Gem::Version.new('3.0')) status = supports_ruby3 ? "✅" : "❌" puts " Ruby 3 support: #{status}" end # Verificar última actualización age = (Date.today - spec.date.to_date).to_i if age > 730 # Más de 2 años puts " ⚠️ Gema antigua (#{age} días sin actualizar)" end puts end
```

4.4 Testing de Compatibilidad

```
# compatibility_test.rb require 'test/unit' class CompatibilityTest < Test::Unit::TestCase def test_ruby_version assert RUBY_VERSION >= "3.0", "Ruby 3.0+ requerido" end def test_removed_constants # Verificar que constantes removidas no se usen refute defined?(Fixnum), "Fixnum removido en Ruby 3" refute defined?(Bignum), "Bignum removido en Ruby 3" end def test_integer_unification # Verificar unificación Integer assert_equal Integer, 1.class assert_equal Integer, (2**100).class end def test_keyword_arguments # Verificar separación de argumentos def test_method(pos, key: nil) [pos, key] end # Esto debe funcionar assert_equal [1, 2], test_method(1, key: 2) # Esto puede generar warning en Ruby 3 # test_method(1, {key: 2}) # Deprecated end def test_hash_syntax # Verificar nueva sintaxis hash old_style = {:key => 'value'} new_style = {key: 'value'} assert_equal old_style, new_style end end
```

🎯 Ejercicio 3: Análisis Completo

Objetivo: Realizar un análisis exhaustivo de tu aplicación.

```
# analysis_script.rb #!/usr/bin/env ruby class MigrationAnalyzer def initialize(project_path) @project_path = project_path @issues = [] end def analyze puts "🔍 Iniciando análisis de migración Ruby 2→3" puts "Proyecto: #{@project_path}" puts "=" * 50 analyze_ruby_files analyze_gemfile analyze_configuration generate_report end private def analyze_ruby_files Dir.glob("#{@project_path}/**/*.rb").each do |file| content = File.read(file) # Análizar patrones problemáticos check_removed_constants(file, content) check_deprecated_methods(file, content) check_hash_syntax(file, content) check_global_variables(file, content) end end def check_removed_constants(file, content) %w[Fixnum Bignum NIL TRUE FALSE].each do |const| if content.match(/\b#{const}\b/) @issues << { file: file, type: :error, message: "Constante removida: #{const}", line: content.lines.find_index { |l| l.include?(const) } + 1 } end end end def check_deprecated_methods(file, content) %w[taint untaint trust untrust].each do |method| if content.match(/\.#{method}\b/) @issues << { file: file, type: :error, message: "Método removido: #{method}", line: content.lines.find_index { |l| l.include?(".#{method}") } + 1 } end end end def check_hash_syntax(file, content) if content.match(/:(\w+)\s*=>/) @issues << { file: file, type: :warning, message: "Sintaxis hash antigua detectada", suggestion: "Considerar migrar a nueva sintaxis {key: value}" } end end def check_global_variables(file, content) if content.match(/\$[1-9]/) @issues << { file: file, type: :info, message: "Uso de variables globales $1, $2, etc.", suggestion: "Verificar comportamiento en Ruby 3" } end end def analyze_gemfile # Implementar análisis de Gemfile end def analyze_configuration # Implementar análisis de configuración end def generate_report puts "\n📊 REPORTE DE ANÁLISIS" puts "=" * 50 errors = @issues.select { |i| i[:type] == :error } warnings = @issues.select { |i| i[:type] == :warning } infos = @issues.select { |i| i[:type] == :info } puts "❌ Errores: #{errors.count}" puts "⚠️ Advertencias: #{warnings.count}" puts "ℹ️ Información: #{infos.count}" puts "📋 Total: #{@issues.count}" if errors.any? puts "\n❌ ERRORES CRÍTICOS:" errors.each do |issue| puts "  📁 #{issue[:file]}" puts "  #{issue[:message]}" puts "  Línea: #{issue[:line]}" if issue[:line] puts end end end end # Usar el analizador if ARGV[0] analyzer = MigrationAnalyzer.new(ARGV[0]) analyzer.analyze else puts "Uso: ruby analysis_script.rb /ruta/al/proyecto" end
```

Resultado esperado: Un reporte detallado de todos los problemas de compatibilidad encontrados.

5. □ Constantes Removidas

Unificación del Sistema Numérico

Ruby 3 elimina varias constantes que fueron unificadas en versiones anteriores, siendo la más importante la unificación de Fixnum y Bignum en Integer.

5.1 Fixnum y Bignum → Integer

□ Ruby 2 (Problemático)

```
# Verificación de tipo antigua if number.class == Fixnum puts "Número pequeño" elsif number.class == Bignum puts "Número grande" end # Case statement problemático case number.class when Fixnum handle_small_number(number) when Bignum handle_big_number(number) end # Constantes en comparaciones def is_integer?(obj) obj.class == Fixnum || obj.class == Bignum end
```

□ Ruby 3 (Correcto)

```
# Verificación unificada if number.class == Integer puts "Número entero" end # Mejor: usar is_a? if number.is_a?(Integer) puts "Es un entero" end # Case statement corregido case number.class when Integer handle_integer(number) end # Función corregida def is_integer?(obj) obj.is_a?(Integer) end # Alternativa más idiomática def is_integer?(obj) obj.kind_of?(Integer) end
```

5.2 Constantes Booleanas: NIL, TRUE, FALSE

□ Ruby 2 (Problemático)

```
# Uso de constantes removidas value = NIL result = TRUE error_state = FALSE # En comparaciones if response == NIL handle_nil_response end # En case statements case status when TRUE process_success when FALSE process_failure when NIL process_unknown end # Asignaciones directas DEFAULT_VALUE = NIL SUCCESS_STATUS = TRUE FAILURE_STATUS = FALSE
```

✅ Ruby 3 (Correcto)

```
# Usar literales directos value = nil result = true error_state = false # Comparaciones corregidas if response.nil? handle_nil_response end # Case statements corregidos case status when true process_success when false process_failure when nil process_unknown end # Constantes corregidas DEFAULT_VALUE = nil SUCCESS_STATUS = true FAILURE_STATUS = false # Mejor práctica: usar símbolos DEFAULT_VALUE = :none SUCCESS_STATUS = :success FAILURE_STATUS = :failure
```

5.3 Herramientas de Migración Automática

```
#!/usr/bin/env ruby # constant_migrator.rb - Migrador automático de constantes class ConstantMigrator CONSTANT_MAPPINGS = { 'Fixnum' => 'Integer', 'Bignum' => 'Integer', 'NIL' => 'nil', 'TRUE' => 'true', 'FALSE' => 'false' }.freeze def initialize(file_path) @file_path = file_path @content = File.read(file_path) @changes_made = [] end def migrate! backup_file CONSTANT_MAPPINGS.each do |old_const, new_const| migrate_constant(old_const, new_const) end write_changes if @changes_made.any? report_changes end private def migrate_constant(old_const, new_const) pattern = /\b#{Regexp.escape(old_const)}\b/ matches = @content.scan(pattern) if matches.any? @content.gsub!(pattern, new_const) @changes_made << { old: old_const, new: new_const, count: matches.size } end end def backup_file backup_name = "#{@file_path}.ruby2.backup" File.write(backup_name, File.read(@file_path)) puts "📁 Backup creado: #{backup_name}" end def write_changes File.write(@file_path, @content) puts "💾 Archivo actualizado: #{@file_path}" end def report_changes if @changes_made.any? puts "📊 Cambios realizados:" @changes_made.each do |change| puts " • #{change[:old]} → #{change[:new]} (#{change[:count]} ocurrencias)" end else puts "ℹ️ No se encontraron constantes a migrar" end end end # Uso del migrador if ARGV.empty? puts "Uso: ruby constant_migrator.rb archivo.rb" exit 1 end ARGV.each do |file| if File.exist?(file) puts "🔄 Migrando: #{file}" migrator = ConstantMigrator.new(file) migrator.migrate! puts else puts "❌ Archivo no encontrado: #{file}" end end
```

5.4 Testing de Constantes

```
# test/test_constants.rb require 'test/unit' class ConstantMigrationTest < Test::Unit::TestCase def test_fixnum_bignum_removed # Verificar que las constantes no existen assert_raises(NameError) { Fixnum } assert_raises(NameError) { Bignum } puts "✅ Fixnum y Bignum correctamente removidos" end def test_integer_unification small_number = 1 big_number = 2**100 # Ambos deben ser Integer assert_equal Integer, small_number.class assert_equal Integer, big_number.class # Verificar que is_a? funciona assert small_number.is_a?(Integer) assert big_number.is_a?(Integer) puts "✅ Unificación Integer funcionando" end def test_boolean_constants_removed # Verificar que las constantes booleanas no existen assert_raises(NameError) { NIL } assert_raises(NameError) { TRUE } assert_raises(NameError) { FALSE } puts "✅ Constantes booleanas correctamente removidas" end def test_literal_values_work # Verificar que los literales funcionan assert_nil nil assert_equal true, true assert_equal false, false # Verificar comparaciones assert nil.nil? assert_not true.nil? assert_not false.nil? puts "✅ Literales booleanos funcionando" end end # Ejecutar tests if __FILE__ == $0 puts "🚀 Ejecutando tests de constantes..." # Los tests se ejecutarán automáticamente end
```

🎯 Ejercicio 4: Migración de Constantes

Objetivo: Migrar todas las constantes removidas en un proyecto real.

```
# migration_exercise.rb class ConstantMigrationExercise def self.run puts "🎯 Ejercicio: Migración de Constantes" puts "=" * 40 # Crear archivos de ejemplo con problemas create_sample_files # Ejecutar migración migrate_all_files # Verificar resultados verify_migration puts "🎉 Ejercicio completado" end def self.create_sample_files sample_code = <<~RUBY class ExampleClass def check_number_type(num) case num.class when Fixnum "Small number: \#{num}" when Bignum "Big number: \#{num}" else "Not a number" end end def process_value(val) return NIL if val.nil? return TRUE if val == 1 return FALSE if val == 0 val end end RUBY File.write('example_class.rb', sample_code) puts "📝 Archivo de ejemplo creado: example_class.rb" end def self.migrate_all_files Dir.glob('*.rb').each do |file| next if file == __FILE__ migrator = ConstantMigrator.new(file) migrator.migrate! end end def self.verify_migration puts "\n🔍 Verificando migración..." Dir.glob('*.rb').each do |file| next if file == __FILE__ || file.include?('.backup') content = File.read(file) # Verificar que no quedan constantes problemáticas problems = [] %w[Fixnum Bignum NIL TRUE FALSE].each do |const| problems << const if content.include?(const) end if problems.empty? puts "✅ #{file}: Sin problemas" else puts "⚠️ #{file}: Constantes pendientes: #{problems.join(', ')}" end end end end # Ejecutar ejercicio ConstantMigrationExercise.run if __FILE__ == $0
```

Resultado esperado: Todos los archivos migrados sin constantes problemáticas y con backups creados.

6. Métodos Deprecados y Eliminados

Limpieza del API de Ruby

Ruby 3 elimina varios métodos que fueron deprecados en versiones anteriores, principalmente relacionados con el sistema de "taint" y "trust".

6.1 Sistema Taint/Trust Eliminado

⚠ **Métodos Completamente Removidos en Ruby 3:**

- Object#taint
- Object#untaint
- Object#tainted?
- Object#trust
- Object#untrust
- Object#untrusted?

Ruby 2 (Removido)

```
# Sistema taint para seguridad user_input = gets.chomp user_input.taint # Marcar como no confiable # Verificar estado taint if user_input.tainted? sanitize_input(user_input) end # Remover taint después de sanitizar clean_input = sanitize_input(user_input) clean_input.untaint # Sistema trust data = external_source.fetch data.untrust # Marcar como no confiable if data.untrusted? validate_data(data) end # Confiar en datos validados validated_data.trust
```

Ruby 3 (Alternativas)

```
# Usar validación explícita user_input = gets.chomp # Mantener estado de validación manualmente class SecureString attr_reader :value, :validated def initialize(value) @value = value @validated = false end def validate! # Lógica de validación @value = sanitize(@value) @validated = true self end def safe_value raise SecurityError unless @validated @value end end # Uso del sistema personalizado secure_input = SecureString.new(user_input) secure_input.validate! safe_data = secure_input.safe_value # Alternativa con módulos module SecurityMarker def mark_untrusted @untrusted = true end def mark_trusted @untrusted = false end def trusted? !@untrusted end end
```

6.2 Otros Métodos Deprecados

Método Removido	Clase	Alternativa en Ruby 3	Notas
Dir.exists?	Dir	Dir.exist?	Cambio de nombre
File.exists?	File	File.exist?	Cambio de nombre
Integer#size	Integer	No hay reemplazo directo	Poco usado
Proc#==	Proc	Proc#equal?	Cambio semántico

6.3 Migración Automática de Métodos

```
#!/usr/bin/env ruby # method_migrator.rb - Migrador de métodos deprecados class MethodMigrator REMOVED_METHODS = [ 'taint', 'untaint', 'tainted?', 'trust', 'untrust', 'untrusted?' ].freeze METHOD_REPLACEMENTS = { 'File.exists?' => 'File.exist?', 'Dir.exists?' => 'Dir.exist?' }.freeze def initialize(file_path) @file_path = file_path @content = File.read(file_path) @issues = [] end def analyze_and_migrate! puts "🔍 Analizando: #{@file_path}" find_removed_methods apply_replacements if @issues.any? create_backup generate_migration_report write_changes else puts "✅ No se encontraron métodos a migrar" end end private def find_removed_methods REMOVED_METHODS.each do |method| pattern = /\.#{method}(\b|\()/ @content.scan(pattern) do |match| line_number = @content[0..$~.begin(0)].count("\n") + 1 @issues << { type: :removed_method, method: method, line: line_number, action: :manual_review_required } end end end def apply_replacements METHOD_REPLACEMENTS.each do |old_method, new_method| if @content.include?(old_method) @content.gsub!(old_method, new_method) @issues << { type: :method_replacement, old: old_method, new: new_method, action: :automatically_replaced } end end end def create_backup backup_path = "#{@file_path}.pre_method_migration.backup" File.write(backup_path, File.read(@file_path)) puts "💾 Backup creado: #{backup_path}" end def generate_migration_report puts "\n📊 REPORTE DE MIGRACIÓN DE MÉTODOS" puts "=" * 50 automatic_changes = @issues.select { |i| i[:action] == :automatically_replaced } manual_reviews = @issues.select { |i| i[:action] == :manual_review_required } if automatic_changes.any? puts "✅ Cambios automáticos aplicados:" automatic_changes.each do |change| puts " • #{change[:old]} → #{change[:new]}" end puts end if manual_reviews.any? puts "⚠️ Revisión manual requerida:" manual_reviews.each do |issue| puts " • Línea #{issue[:line]}: Método removido '#{issue[:method]}'" puts " Acción: Implementar alternativa de seguridad" end puts generate_security_suggestions(manual_reviews) end end def generate_security_suggestions(manual_reviews) puts "🔒 SUGERENCIAS PARA MÉTODOS DE SEGURIDAD:" puts "=" * 50 security_methods = manual_reviews.select { |i| %w[taint untaint tainted? trust untrust untrusted?].include?(i[:method]) } if security_methods.any? puts <<~SUGGESTIONS Para reemplazar el sistema taint/trust, considera: 1. Validación Explícita: ```ruby class SecureData def initialize(value) @value = value @validated = false end def validate! # Tu lógica de validación @validated = true end def safe_value raise SecurityError unless @validated @value end end ``` 2. Usar bibliotecas de sanitización: - Loofah para HTML - Addressable para URLs - ActiveRecord para SQL 3. Patrones de validación: - Whitelisting en lugar de blacklisting - Validación en la entrada y salida - Uso de tipos seguros SUGGESTIONS end end def write_changes File.write(@file_path, @content) puts "💾 Cambios aplicados a: #{@file_path}" end end # Script principal if ARGV.empty? puts "Uso: ruby method_migrator.rb archivo.rb [archivo2.rb ...]" puts "O para migrar todo un directorio:" puts "find . -name '*.rb' -exec ruby method_migrator.rb {} +" exit 1 end ARGV.each do |file_path| if File.exist?(file_path) && file_path.end_with?('.rb') migrator = MethodMigrator.new(file_path) migrator.analyze_and_migrate! puts else puts "⚠️ Saltando: #{file_path} (no es archivo Ruby válido)" end end
```

6.4 Alternativas de Seguridad

```
# security_alternatives.rb - Alternativas al sistema taint/trust # 1. Sistema de validación basado en clases class ValidatedString attr_reader :raw_value def initialize(value) @raw_value = value.to_s @validations = [] @sanitizers = [] end def add_validation(&block) @validations << block self end def add_sanitizer(&block) @sanitizers << block self end def validate! @validations.each do |validation| raise SecurityError, "Validation failed" unless validation.call(@raw_value) end self end def sanitize! @sanitizers.each do |sanitizer| @raw_value = sanitizer.call(@raw_value) end self end def safe_value validate! sanitize! @raw_value end end # 2. Módulo para marcar objetos module SecurityMarker def self.included(base) base.extend(ClassMethods) end module ClassMethods def secure_attr_accessor(*attrs) attrs.each do |attr| define_method(attr) do instance_variable_get("@#{attr}") end define_method("#{attr}=") do |value| if value.is_a?(String) && !value.frozen? warn "Warning: Setting mutable string to secure attribute" end instance_variable_set("@#{attr}", value) end define_method("#{attr}_validated?") do instance_variable_get("@#{attr}_validated") || false end define_method("validate_#{attr}!") do |&block| value = instance_variable_get("@#{attr}") if block_given? raise SecurityError unless block.call(value) end instance_variable_set("@#{attr}_validated", true) end end end end # 3. Factory para objetos seguros class SecureObjectFactory def self.create_secure_string(value) ValidatedString.new(value) .add_validation { |v| v.length < 1000 } .add_validation { |v| !v.include?(']/, '') } end def self.create_secure_email(email) ValidatedString.new(email) .add_validation { |e| e.match?(/\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i) } .add_sanitizer { |e| e.strip.downcase } end def self.create_secure_url(url) require 'uri' ValidatedString.new(url) .add_validation { |u| URI.parse(u) rescue false } .add_validation { |u| !u.match?(/javascript:|data:/i) } .add_sanitizer { |u| URI.parse(u).to_s } end end # Ejemplos de uso puts "🔒 Ejemplos de Sistema de Seguridad Alternativo" puts "=" * 50 # Ejemplo 1: String validado user_input = "Hello World" secure_string = SecureObjectFactory.create_secure_string(user_input) begin safe_output = secure_string.safe_value puts "✅ String seguro: '#{safe_output}'" rescue SecurityError => e puts "❌ Error de seguridad: #{e.message}" end # Ejemplo 2: Email validado email_input = "USER@EXAMPLE.COM " secure_email = SecureObjectFactory.create_secure_email(email_input) safe_email = secure_email.safe_value puts "✅ Email seguro: '#{safe_email}'" # Ejemplo 3: Con módulo de seguridad class User include SecurityMarker secure_attr_accessor :name, :email def initialize(name, email) self.name = name self.email = email end def display_info validate_name! { |n| n.length > 0 && n.length < 100 } validate_email! { |e| e.match?(/\A[\w+\-.]+@[a-z\d\-]+(\.[a-z\d\-]+)*\.[a-z]+\z/i) } "User: #{name} (#{email})" end end user = User.new("John Doe", "john@example.com") puts "✅ #{user.display_info}"
```

🎯 Ejercicio 5: Migración de Métodos de Seguridad

Objetivo: Reemplazar métodos taint/trust con sistema de seguridad moderno.

```
# exercise_security_migration.rb # Código problemático a migrar class LegacySecuritySystem def process_user_input(input) # Código Ruby 2 con taint input.taint if input.tainted? cleaned = sanitize_input(input) cleaned.untaint return cleaned end input end def handle_external_data(data) data.untrust if data.untrusted? validated = validate_data(data) validated.trust return validated end data end private def sanitize_input(input) input.gsub(/[<>]/, '') end def validate_data(data) # Simulación de validación data.strip end end # TU TAREA: Implementar ModernSecuritySystem class ModernSecuritySystem # TODO: Implementar reemplazo para process_user_input # TODO: Implementar reemplazo para handle_external_data # TODO: Usar el patrón ValidatedString o crear tu propio sistema def process_user_input(input) # Tu implementación aquí end def handle_external_data(data) # Tu implementación aquí end end # Test de la implementación def test_security_systems puts "🔒 Testing Security Systems" puts "=" * 30 test_input = "Hello" test_data = " untrusted data " # Sistema legacy (no funcionará en Ruby 3) begin legacy = LegacySecuritySystem.new legacy.process_user_input(test_input) puts "⚠️ Legacy system should not work in Ruby 3" rescue NoMethodError => e puts "✅ Legacy system correctly fails: #{e.message}" end # Tu sistema moderno modern = ModernSecuritySystem.new # Implementar tests para tu sistema # processed = modern.process_user_input(test_input) # handled = modern.handle_external_data(test_data) puts "TODO: Implementar y probar tu sistema de seguridad" end # Ejecutar test test_security_systems if __FILE__ == $0
```

Resultado esperado: Sistema de seguridad moderno que reemplaza completamente el sistema taint/trust removido.

7. □ Cambios de Sintaxis

Modernización de la Sintaxis Ruby

Ruby 3 incluye varios cambios en la sintaxis para hacer el código más consistente y expresivo, siendo el más notable el cambio en la sintaxis de hash.

7.1 Sintaxis de Hash: Rockets vs Colons

⚠ Ruby 2 (Estilo Antiguo)

```
# Hash rockets (funciona pero es estilo viejo) user = { :name => 'John Doe', :email => 'john@example.com', :age => 30, :active => true } # Mezclado (inconsistente) config = { :host => 'localhost', :port => 3000, 'timeout' => 30, :ssl => false } # En llamadas a métodos User.create( :name => 'Jane', :email => 'jane@example.com' ) # Hash anidados response = { :status => 'success', :data => { :user => { :id => 1, :name => 'John' } } }
```

□ Ruby 3 (Estilo Moderno)

```
# Sintaxis moderna con colons user = { name: 'John Doe', email: 'john@example.com', age: 30, active: true } # Consistente con strings como keys config = { host: 'localhost', port: 3000, 'timeout' => 30, # String key mantiene => ssl: false } # En llamadas a métodos (más limpio) User.create( name: 'Jane', email: 'jane@example.com' ) # Hash anidados consistentes response = { status: 'success', data: { user: { id: 1, name: 'John' } } } # Mixing symbols and strings cuando necesario mixed_hash = { symbol_key: 'value', 'string_key' => 'value', 123 => 'numeric_key' # Non-symbol keys use => }
```

7.2 Herramienta de Conversión de Sintaxis Hash

```
#!/usr/bin/env ruby # hash_syntax_converter.rb - Convertir sintaxis hash a estilo moderno class HashSyntaxConverter def initialize(file_path) @file_path = file_path @content = File.read(file_path) @conversions = 0 end def convert! puts "🔄 Convirtiendo sintaxis hash: #{@file_path}" original_content = @content.dup # Patrón para encontrar hash rockets con símbolos # :symbol => value se convierte a symbol: value pattern = /:(\w+)\s*=>\s*/ @content.gsub!(pattern) do |match| symbol_name = $1 @conversions += 1 "#{symbol_name}: " end if @conversions > 0 create_backup write_changes puts "✅ #{@conversions} conversiones aplicadas" else puts "ℹ️ No se encontraron hash rockets a convertir" end end def preview_changes puts "👁️ Vista previa de cambios:" puts "─" * 50 lines = @content.lines pattern = /:(\w+)\s*=>\s*/ lines.each_with_index do |line, index| if line.match(pattern) old_line = line.dup new_line = line.gsub(pattern) { |match| "#{$1}: " } puts "Línea #{index + 1}:" puts "  - #{old_line.strip}" puts "  + #{new_line.strip}" puts end end end private def create_backup backup_path = "#{@file_path}.hash_syntax.backup" File.write(backup_path, File.read(@file_path)) puts "💾 Backup creado: #{backup_path}" end def write_changes File.write(@file_path, @content) puts "📝 Archivo actualizado" end end # Convertidor en lote class BatchHashConverter def self.convert_directory(dir_path, options = {}) preview_only = options[:preview] || false puts "🔄 Conversión masiva de sintaxis hash" puts "Directorio: #{dir_path}" puts "Modo: #{preview_only ? 'Vista previa' : 'Aplicar cambios'}" puts "=" * 50 pattern = File.join(dir_path, '**', '*.rb') total_files = 0 total_conversions = 0 Dir.glob(pattern).each do |file_path| next if file_path.include?('.backup') converter = HashSyntaxConverter.new(file_path) if preview_only converter.preview_changes else original_conversions = converter.instance_variable_get(:@conversions) converter.convert! total_conversions += converter.instance_variable_get(:@conversions) end total_files += 1 end puts "📊 RESUMEN:" puts "Archivos procesados: #{total_files}" puts "Conversiones totales: #{total_conversions}" unless preview_only end end # Script principal if ARGV.empty? puts <<~USAGE Uso: ruby hash_syntax_converter.rb archivo.rb # Convertir archivo individual ruby hash_syntax_converter.rb --dir directorio # Convertir directorio ruby hash_syntax_converter.rb --preview archivo # Vista previa ruby hash_syntax_converter.rb --batch directorio # Conversión masiva USAGE exit 1 end case ARGV[0] when '--dir', '--batch' dir_path = ARGV[1] || '.' BatchHashConverter.convert_directory(dir_path) when '--preview' file_path = ARGV[1] if File.exist?(file_path) converter = HashSyntaxConverter.new(file_path) converter.preview_changes else puts "❌ Archivo no encontrado: #{file_path}" end else file_path = ARGV[0] if File.exist?(file_path) converter = HashSyntaxConverter.new(file_path) converter.convert! else puts "❌ Archivo no encontrado: #{file_path}" end end
```

7.3 Cambios en String Literals

⚠ Ruby 2 (Comportamiento Inconsistente)

```
# String literals eran mutables por defecto str1 = "Hello" str2 = "Hello" str1.object_id != str2.object_id # true, objetos diferentes # Problema de performance def get_message "Static message" # Nuevo objeto cada llamada end 1000.times { get_message } # 1000 objetos creados # Congelado manual CONSTANT_STRING = "Immutable".freeze
```

✅ Ruby 3 (Frozen String Literals)

```
# magic_comment para strings inmutables # frozen_string_literal: true str1 = "Hello" str2 = "Hello" str1.object_id == str2.object_id # true, mismo objeto # Mejor performance automática def get_message "Static message" # Reutiliza mismo objeto end 1000.times { get_message } # Solo 1 objeto # Strings inmutables por defecto message = "Immutable by default" # message << " more text" # Error: FrozenError # String mutable cuando necesario mutable = +"Mutable string" mutable << " can be modified" # O usar String.new mutable2 = String.new("Also mutable")
```

7.4 Mejoras en Pattern Matching (Ruby 3.0+)

```
# pattern_matching_examples.rb - Nuevas características Ruby 3 # Pattern matching básico def describe_data(data) case data in { type: 'user', name: String => name, age: Integer => age } "User #{name} is #{age} years old" in { type: 'product', name: String => name, price: Numeric => price } "Product #{name} costs $#{price}" in { type: 'error', message: String => msg } "Error: #{msg}" else "Unknown data format" end end # Array pattern matching def process_coordinates(coords) case coords in [x, y] if x.is_a?(Numeric) && y.is_a?(Numeric) "2D point: (#{x}, #{y})" in [x, y, z] if [x, y, z].all? { |n| n.is_a?(Numeric) } "3D point: (#{x}, #{y}, #{z})" in [] "Empty coordinates" else "Invalid coordinate format" end end # Guard clauses en pattern matching def categorize_number(num) case num in Integer => n if n < 0 "Negative integer: #{n}" in Integer => n if n == 0 "Zero" in Integer => n if n > 0 && n <= 10 "Small positive integer: #{n}" in Integer => n if n > 10 "Large positive integer: #{n}" in Float => f "Floating point: #{f}" else "Not a number" end end # Ejemplos de uso puts "🎯 Pattern Matching Examples" puts "=" * 30 # Test describe_data user_data = { type: 'user', name: 'Alice', age: 30 } product_data = { type: 'product', name: 'Laptop', price: 999.99 } error_data = { type: 'error', message: 'Connection failed' } puts describe_data(user_data) puts describe_data(product_data) puts describe_data(error_data) # Test process_coordinates puts process_coordinates([1, 2]) puts process_coordinates([1, 2, 3]) puts process_coordinates([]) # Test categorize_number [-5, 0, 7, 15, 3.14].each do |num| puts categorize_number(num) end
```

7.5 Rightward Assignment (Ruby 3.0+)

```
# rightward_assignment.rb - Nueva sintaxis de asignación # Asignación tradicional traditional_result = complex_calculation(data) process_result(traditional_result) # Rightward assignment (Ruby 3.0+) complex_calculation(data) => result process_result(result) # Útil en pattern matching def process_json_response(json_string) JSON.parse(json_string) => { 'status' => status, 'data' => data } case status when 'success' handle_success(data) when 'error' handle_error(data) end end # En one-liners def quick_process(items) items.map(&:to_i).select(&:positive?) => positive_numbers positive_numbers.sum end # Combinado con case/in def analyze_request(request) case request in { method: 'GET', path: String } => req handle_get_request(req) in { method: 'POST', body: Hash } => req handle_post_request(req) end end # Ejemplos prácticos puts "➡️ Rightward Assignment Examples" # Ejemplo 1: Processing pipeline [1, -2, 3, -4, 5].select(&:positive?) => positive_nums positive_nums.map { |n| n * 2 } => doubled puts "Positive doubled: #{doubled}" # Ejemplo 2: JSON processing json_data = '{"user": {"name": "John", "age": 30}}' JSON.parse(json_data) => { 'user' => { 'name' => name, 'age' => age } } puts "User: #{name}, Age: #{age}"
```

🔄 Ejercicio 6: Modernización de Sintaxis

Objetivo: Modernizar la sintaxis de un proyecto completo.

```
# syntax_modernization_exercise.rb # Código legacy a modernizar class LegacySyntaxExample def initialize # Hash rockets viejos @config = { :host => 'localhost', :port => 3000, :ssl => false, :timeout => 30 } # String literals mutables @message = "Welcome to our application" # Lógica condicional verbosa @settings = { :theme => 'dark', :language => 'en' } end def process_user_data(data) # Pattern matching verboso con if/elsif if data.is_a?(Hash) && data.has_key?(:type) if data[:type] == 'user' process_user(data) elsif data[:type] == 'admin' process_admin(data) else process_unknown(data) end else raise ArgumentError, "Invalid data format" end end def calculate_total(items) # Lógica tradicional result = items.map { |item| item[:price] * item[:quantity] } total = result.sum apply_discount(total) end private def process_user(data); "Processing user: #{data[:name]}"; end def process_admin(data); "Processing admin: #{data[:name]}"; end def process_unknown(data); "Unknown type: #{data[:type]}"; end def apply_discount(total); total * 0.9; end end # TU TAREA: Crear ModernSyntaxExample class ModernSyntaxExample # TODO: Convertir @config a sintaxis moderna # TODO: Usar frozen_string_literal # TODO: Implementar pattern matching en process_user_data # TODO: Usar rightward assignment en calculate_total def initialize # Tu implementación moderna aquí end def process_user_data(data) # Usar pattern matching case/in end def calculate_total(items) # Usar rightward assignment end end # Tests comparativos def test_syntax_modernization puts "🔄 Testing Syntax Modernization" puts "=" * 35 # Datos de prueba user_data = { type: 'user', name: 'Alice', email: 'alice@example.com' } admin_data = { type: 'admin', name: 'Bob', permissions: ['read', 'write'] } items = [ { price: 10.0, quantity: 2 }, { price: 5.0, quantity: 3 } ] # Test legacy puts "Legacy implementation:" legacy = LegacySyntaxExample.new puts legacy.process_user_data(user_data) puts "Total: $#{legacy.calculate_total(items)}" # Test moderno puts "\nModern implementation:" modern = ModernSyntaxExample.new # TODO: Descomentar cuando implementes # puts modern.process_user_data(user_data) # puts "Total: $#{modern.calculate_total(items)}" puts "\n📝 Completa la implementación ModernSyntaxExample" end # Agregar frozen_string_literal al archivo def add_frozen_string_literal(file_path) content = File.read(file_path) unless content.start_with?("# frozen_string_literal: true") new_content = "# frozen_string_literal: true\n\n#{content}" File.write(file_path, new_content) puts "✅ Added frozen_string_literal to #{file_path}" end end # Ejecutar ejercicio test_syntax_modernization if __FILE__ == $0
```

Resultado esperado: Código completamente modernizado usando todas las nuevas características de sintaxis de Ruby 3.

8. ⚖ Argumentos de Palabras Clave

Separación de Argumentos Posicionales y por Palabra Clave

Ruby 3 introduce una separación estricta entre argumentos posicionales y argumentos por palabra clave, eliminando la conversión automática que existía en Ruby 2.

9. Mejoras de Rendimiento

Ruby 3x3: Objetivo Cumplido

Ruby 3 logra ser significativamente más rápido que Ruby 2, con mejoras en el JIT, garbage collection y optimizaciones de VM.

10. 🛠️ Herramientas de Migración

Ruby Migrator y Herramientas Complementarias

Conjunto completo de herramientas para automatizar y facilitar el proceso de migración.

10.1 Ruby Migrator - Herramienta Principal

```
# Instalación y uso básico gem install ruby_migrator # Análisis completo de proyecto ruby_migrator --path mi_proyecto --report-only --verbose # Migración con respaldo automático ruby_migrator --path mi_proyecto --format json # Análisis específico por tipo ruby_migrator --path app/models --pattern constants
```

15. □ Recursos y Referencias

15.1 Documentación Oficial

- **Ruby 3.0 Release Notes:** https://www.ruby-lang.org/en/news/2020/12/25/ruby-3-0-0-released/
- **Ruby 3.3 Documentation:** https://docs.ruby-lang.org/en/3.3/
- **Migration Guide:** https://www.ruby-lang.org/en/news/2020/09/25/ruby-3-0-0-preview1-released/

15.2 Herramientas de Migración

- **Ruby Migrator:** Herramienta desarrollada en este proyecto
- **RuboCop:** https://rubocop.org/
- **ruby2_keywords:** Gem para compatibilidad de argumentos
- **Bundle Audit:** Análisis de vulnerabilidades en gemas

15.3 Mejores Prácticas

☑ Realizar backup completo antes de migrar

☑ Usar rama separada para migración

☑ Ejecutar suite completa de tests

☐ Migrar dependencias antes que código aplicación

☐ Revisar logs de deprecation warnings

☐ Actualizar CI/CD para Ruby 3

☐ Monitorear rendimiento post-migración

☐ Documentar cambios realizados

□ **¡Felicitaciones!**

Has completado el curso completo de migración de Ruby 2 a Ruby 3. Con los conocimientos y herramientas adquiridas, estás preparado para migrar exitosamente cualquier proyecto Ruby a la versión 3.

Próximos Pasos:

- Aplicar los conocimientos en un proyecto real
- Explorar las nuevas características de Ruby 3.x
- Contribuir al desarrollo de herramientas de migración
- Compartir experiencias con la comunidad Ruby

Migración Ruby 2 → Ruby 3

Ruby Migrator Project | Agosto 2025