MGL7460 — Automne 2016

Programmation Ruby «*in-the-large*» OU Comment développer une application Ruby

Guy Tremblay
Professeur

Département d'informatique
UQAM
http://www.labunix.uqam.ca/~tremblay

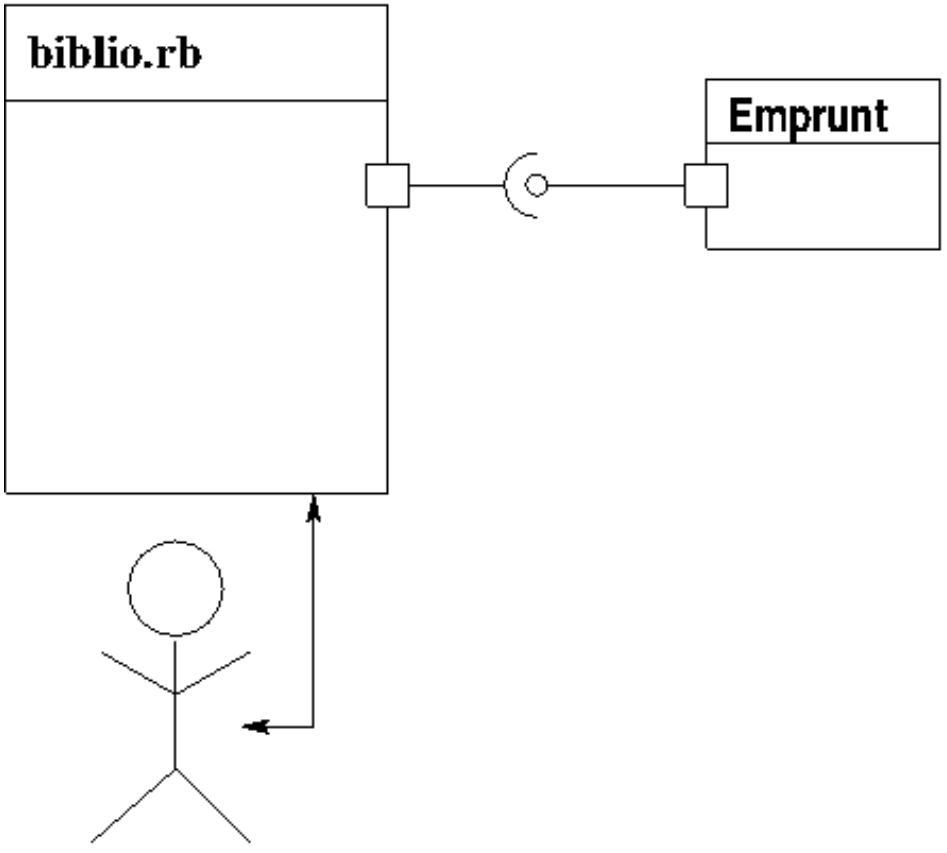
Contenu

1. Introduction/motivation
2. Le choix de la version de Ruby avec `rvm`
3. Les *gems* Ruby
4. Le *gem* `rake` pour l'automatisation des tâches
5. Le *gem* `gli` pour la spécification de suites de commandes
6. Le *gem* `mini-sed`, une version simplifiée de `sed`

1. Introduction/motivation

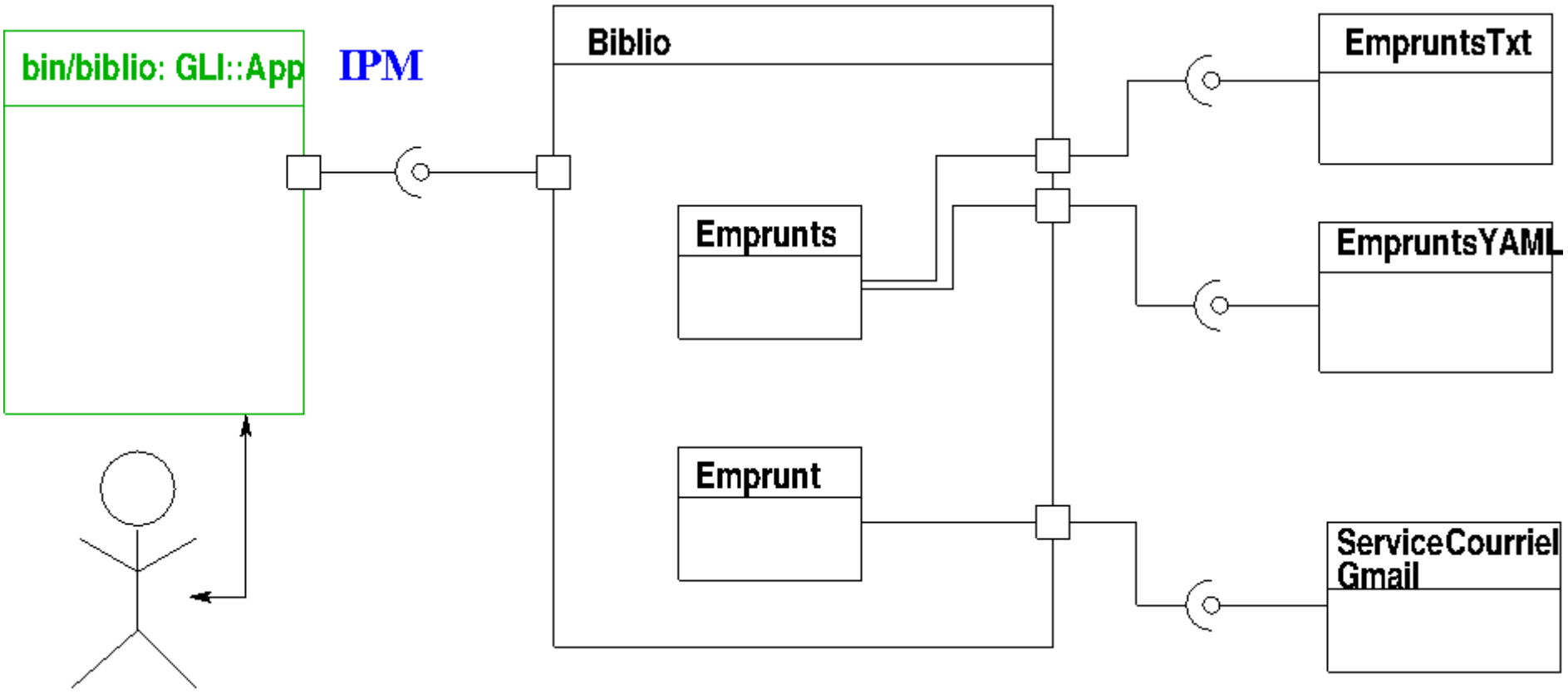
Dans ce qui suit, on va traiter de «programmation-*in-the-large*»

In-the-small



Dans ce qui suit, on va traiter de «programmation-*in-the-large*»

In-the-large



Dans ce qui suit, on va traiter de «programmation-*in-the-large*»

Programming-in-the-small

[P]rogramming in the small describes the activity of writing a small program. Small programs are typified by being small in terms of their source code size, are easy to specify, quick to code and typically perform one task or a few very closely related tasks very well.

`https://en.wikipedia.org/wiki/Programming_in_the_large_and_programming_in_the_small`

Dans ce qui suit, on va traiter de «programmation-*in-the-large*»

⋆

Programming-in-the-large

By large programs we mean systems consisting of many small programs (modules), possibly written by different people. We need languages for programming-in-the-small, i.e. languages not unlike the common programming languages of today, for writing modules. We also need a «module interconnection language» for knitting those modules together into an integrated whole [. . .].

`https://en.wikipedia.org/wiki/Programming_in_the_large_and_programming_in_the_small`

Dans ce qui suit, on va traiter de «programmation-*in-the-large*»

On va le faire en développant... une petite application Ruby en mode «ligne de commandes»

Dans ce qui suit, on va traiter de «programmation-*in-the-large*»

On va le faire en développant... une petite application Ruby en mode «ligne de commandes»

On va aussi examiner certains outils clés de l'écosystème Ruby

2. Le choix de la version de Ruby avec `rvm`

De nombreuses versions du compilateur/interpréteur `Ruby` sont disponibles

```
$ rvm list known
# MRI Rubies
[ruby-]1.8.6[-p420]
[ruby-]1.8.7[-head]
[ruby-]1.9.1[-p431]
[ruby-]1.9.2[-p330]
[ruby-]1.9.3[-p551]
[ruby-]2.0.0[-p643]
[ruby-]2.1.4
[ruby-]2.1[.5]
[ruby-]2.2[.1]
[ruby-]2.2-head
ruby-head

# JRuby
jruby-1.6.8
jruby[-1.7.19]
jruby-head
jruby-9.0.0.0.pre1

# Rubinius
rbx-1.4.3
rbx-2.4.1
rbx[-2.5.2]
rbx-head

# Opal
opal

# Minimalistic ruby implementation - ISO 30170:2012
mruby[-head]

# Ruby Enterprise Edition
ree-1.8.6
ree[-1.8.7][-2012.02]

# GoRuby
goruby

# Topaz
topaz

# MagLev
maglev[-head]
maglev-1.0.0

# Mac OS X Snow Leopard Or Newer
macruby-0.10
macruby-0.11
macruby[-0.12]
macruby-nightly
macruby-head

# IronRuby
ironruby[-1.1.3]
ironruby-head
```

L'outil `rvm` permet d'avoir, sur une même machine ou dans un même compte, différentes versions de Ruby

La commande `rvm list` affiche les versions disponibles

```
$ rvm list

rvm rubies

   jruby-1.7.16.1 [ x86_64 ]
   ruby-1.8.7-head [ x86_64 ]
=> ruby-1.9.3-p550 [ x86_64 ]
 * ruby-2.1.4 [ x86_64 ]
   ruby-2.2.4 [ x86_64 ]

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

$ ruby --version
ruby 1.9.3p550 (2014-10-27 revision 48165) [x86_64-linux
```

L'outil `rvm` permet d'avoir, sur une même machine ou dans un même compte, différentes versions de Ruby

La commande `rvm use` permet de passer d'une version à une autre

```
$ rvm use ruby-2.2.4
Using /home/tremblay/.rvm/gems/ruby-2.2.4

$ ruby --version
ruby 2.2.4p230 (2015-12-16 revision 53155) [x86_64-linux
```

3. Les *gems* Ruby

Qu’est-ce qu’un *gem* Ruby ?

3.1 Qu’est-ce qu’un *gem* Ruby ?

Qu’est-ce qu’un *gem* ?

Un *gem* est un composant logiciel

Dans l’écosystème Ruby, un *gem* est un composant logiciel — une bibliothèque, un paquetage de code — pouvant être utilisé par un autre programme.

Qu’est-ce qu’un *gem* ?

Un *gem* est un composant logiciel

Dans l’écosystème Ruby, un *gem* est un composant logiciel — une bibliothèque, un paquetage de code — pouvant être utilisé par un autre programme.

Un *gem* peut aussi être une application

Un *gem* peut aussi fournir une application, par exemple, une application utilisable par l’intermédiaire de la ligne de commandes.

Un *gem* est un composant logiciel – pouvant être distribué et installé

La commande `gem list` affiche les `gems` installés avec leur numéro de version

```
$ gem list

*** LOCAL GEMS ***

aruba (0.13.0)
bigdecimal (1.2.6)
builder (3.2.2)
bundle (0.0.1)
bundler (1.11.2)
bundler-unload (1.0.2)
childprocess (0.5.9)
contracts (0.13.0)
cucumber (2.3.2)
cucumber-core (1.4.0)
cucumber-wire (0.0.1)
diff-lcs (1.2.5)
executable-hooks (1.3.2)
ffi (1.9.10)
gem-wrappers (1.2.7)
gherkin (3.2.0)
io-console (0.4.3)
json (1.8.1)
multi_json (1.11.2)
multi_test (0.1.2)
psych (2.0.8)
rake (10.4.2)
rdoc (4.2.0)
rspec-expectations (3.4.0)
rspec-support (3.4.1)
rubygems-bundler (1.4.4)
rvm (1.11.3.9)
thor (0.19.1)
```

Un *gem* est un composant logiciel pouvant être distribué et installé

La commande `gem install` permet d'obtenir et d'installer, «à bras», un *gem*

Installation de la dernière version d'un gem

```
$ gem install gli
Fetching: gli-2.13.4.gem (100%)
Successfully installed gli-2.13.4
Parsing documentation for gli-2.13.4
Installing ri documentation for gli-2.13.4
Done installing documentation for gli after 1 seconds
1 gem installed
```

Un *gem* est un composant logiciel pouvant être distribué et installé

La commande `gem install` permet d'obtenir et d'installer, «à bras», un *gem*

Installation de la dernière version d'un gem

```
$ gem install gli
Fetching: gli-2.13.4.gem (100%)
Successfully installed gli-2.13.4
Parsing documentation for gli-2.13.4
Installing ri documentation for gli-2.13.4
Done installing documentation for gli after 1 seconds
1 gem installed
```

Installation d'une version spécifique d'un gem

```
$ gem install gli -v 2.11.0
Fetching: gli-2.11.0.gem (100%)
Successfully installed gli-2.11.0
Parsing documentation for gli-2.11.0
Installing ri documentation for gli-2.11.0
Done installing documentation for gli after 0 seconds
1 gem installed
```

3.2 Comment utiliser un *gem*

Certains gems sont des applications

```
$ gem list | grep rake
rake (11.1.1, 10.4.2, 10.3.2, 10.1.0, 0.9.6)

$ rake --version
rake, version 11.1.1
```

Certains gems sont des composants logiciels qu'on peut utiliser dans un programme avec `require`

```
$ gem list | grep pruby
pruby (0.1.1)

$ pruby
bash: pruby: commande inconnue...
Commande similaire: 'ruby'

$ irb
>> require 'pruby'
=> true

>> PRuby::VERSION
=> "0.1.1"
```

Certains gems sont des composants logiciels qu'on peut utiliser dans un programme avec `require`

```
$ gem list | grep pruby
pruby (0.1.1)

$ pruby
bash: pruby: commande inconnue...
Commande similaire: 'ruby'

$ irb
>> require 'pruby'
=> true

>> PRuby::VERSION
=> "0.1.1"
```

Note : Et certains *gems* sont à la fois composants (utilisables via `require`) et applications (utilisables comme programmes).

L'instruction `require` charge le code d'un *gem* . . . et n'effectue ce chargement qu'une seule fois

```
$ irb
>> require 'pruby'
=> true

>> PRuby::VERSION
=> "0.1.1"

>> require 'pruby'
=> false

>> PRuby::VERSION
=> "0.1.1"
```

Note : Il existe aussi une instruction `load`, qu'on utilise rarement, et qui charge le fichier même si déjà chargé

L'instruction `require` utilise par défaut la version (installée) la plus récente du *gem*

```
$ gem list | grep rake
rake (11.1.1, 10.4.2, 10.3.2, 10.1.0, 0.9.6)

$ irb
>> require 'rake'
=> true

>> Rake::VERSION
=> "11.1.1"
```

L'instruction `gem` utilisée avant `require` permet de spécifier une version spécifique à utiliser

```
$ gem list | grep rake
rake (11.1.1, 10.4.2, 10.3.2, 10.1.0, 0.9.6)

$ irb
>> gem 'rake', '=10.4.2'
=> true

>> require 'rake'
=> true

>> Rake::VERSION
=> "10.4.2"
```

Si un *gem* n'est pas installé on ne peut évidemment pas l'utiliser avec `require`

```
$ gem list | grep pruby

$ pruby
bash: pruby : commande introuvable

$ irb
>> require 'pruby'
LoadError: cannot load such file -- pruby
  from /home/tremblay/.rvm/rubies/ruby-2.1.4/lib/
       ruby/site_ruby/2.1.0/rubygems/core_ext/
       kernel_require.rb:54:in 'require'
  from /home/tremblay/.rvm/rubies/ruby-2.1.4/lib/
       ruby/site_ruby/2.1.0/rubygems/core_ext/
       kernel_require.rb:54:in 'require'
  from (irb):1
  from /home/tremblay/.rvm/rubies/ruby-2.1.4/bin/
       irb:11:in '<main>'
```

3.3 Comment spécifier les *gems* dont on a besoin

Lorsqu'on veut créer un *gem* pour un projet `foo`, on utilise (typiquement) une structure standard de répertoires

```
$ tree foo
foo
|-- bin
|   `-- foo
|-- features
|   |-- foo.feature
|
|-- step_definitions
|   |
`-- foo_steps.rb
|   `-- support
|       `-- env.rb
|-- foo.gemspec
|-- foo.rdoc
|-- Gemfile
|-- lib
|   |-- foo
|   |   `-- version.rb
|   `-- foo.rb
|-- Rakefile
|-- README.rdoc
`-- test
    |-- default_test.rb
    `-- test_helper.rb

7 directories, 13 files
```

Le fichier `foo.gemspec` décrit alors les méta-informations du *gem* ainsi que ses dépendances

Dépendances = quels gems sont nécessaires pour que `foo` fonctionne

```
$ cat foo.gemspec
require File.join([File.dirname(__FILE__),'lib','foo','version
spec = Gem::Specification.new do |s|
  s.name = 'foo'
  s.version = Foo::VERSION
  s.author = 'Guy Tremblay'
  s.email = tremblay.guy@uqam.ca
  s.homepage = 'http://www.labunix.uqam.ca/~tremblay'
  s.platform = Gem::Platform::RUBY
  s.summary = 'Mon projet foo'
  s.files = `git ls-files`.split("")
  s.require_paths << 'lib'
  s.bindir = 'bin'
  s.executables << 'foo'

  s.add_development_dependency('rake')
  s.add_development_dependency('rdoc')
  s.add_development_dependency('aruba')
  s.add_runtime_dependency('gli','2.12.0')
end
```

De nombreux outils génèrent automatiquement la structure de répertoires et le fichier gemspec requis

Exemple : L'outil `gli`, pour définir des interfaces personne-machine «à la `git`»

```
$ gli --help
NAME
    gli - create scaffolding for a GLI-powered application

SYNOPSIS
    gli [global options] command [command options] [arguments.

VERSION
    2.12.0

GLOBAL OPTIONS
    --help             - Show this message
    -n                 - Dry run; dont change the disk
    -r, --root=arg     - Root dir of project (default: .)
    -v                 - Be verbose
    --version          - Display the program version

COMMANDS
    help               - Shows a list of commands or help for one
    init, scaffold     - Create a new GLI-based project
```

De nombreux outils génèrent automatiquement la structure de répertoires et le fichier gemspec requis

Exemple : L'outil `gli`, pour définir des interfaces personne-machine «à la `git`»

```
$ gli init bar
Creating dir ./bar/lib...
Creating dir ./bar/bin...
Creating dir ./bar/test...
Created ./bar/bin/bar
Created ./bar/README.rdoc
Created ./bar/bar.rdoc
Created ./bar/bar.gemspec
Created ./bar/test/default_test.rb
Created ./bar/test/test_helper.rb
Created ./bar/Rakefile
Created ./bar/Gemfile
Created ./bar/features
Created ./bar/lib/bar/version.rb
Created ./bar/lib/bar.rb
```

De nombreux outils génèrent automatiquement la structure de répertoires et le fichier gemspec requis

Exemple : L'outil `gli`, pour définir des interfaces personne-machine «à la `git`»

```
$ cat bar/bar.gemspec
# Ensure we require the local version and not one we might have installed already
require File.join([File.dirname(__FILE__),'lib','bar','version
spec = Gem::Specification.new do |s|
  s.name = 'bar'
  s.version = Bar::VERSION
  s.author = 'Your Name Here'
  s.email = 'your@email.address.com'
  s.homepage = 'http://your.website.com'
  s.platform = Gem::Platform::RUBY
  s.summary = 'A description of your project'
  s.files = `git ls-files`.split("")
  s.require_paths << 'lib'
  s.has_rdoc = true
  s.extra_rdoc_files = ['README.rdoc','bar.rdoc']
  s.bindir = 'bin'
  s.executables << 'bar'
  s.add_development_dependency('rake')
  ...
  s.add_runtime_dependency('gli','2.12.0')
end
```

La spécification des dépendances dans un `.gemspec`… ne rend pas les *gems* disponibles

- Le fichier `.gemspec` indique, de façon claire et explicite, les dépendances, i.e., les *gems* requis.

  Mais… il faut ensuite s'assurer que ces *gems* sont effectivement installés !

- Deux approches possibles :
  - Manuellement — «à bras» ☹
  - Automatiquement — «à l'aide d'un outil» ☺

La spécification des dépendances dans un `.gemspec`... ne rend pas les *gems* disponibles

- Le fichier `.gemspec` indique, de façon claire et explicite, les dépendances, i.e., les *gems* requis.

  Mais... il faut ensuite s'assurer que ces *gems* sont effectivement installés !

- Deux approches possibles :
  - Manuellement — «à bras» ☹
  - Automatiquement — «à l'aide d'un outil» ☺

  `bundler`

3.4 L’outil `bundler`

L'outil `bundler` permet d'installer automatiquement des *gems* à partir d'un `Gemfile`

Les gems requis sont spécifiés dans le fichier `Gemfile`

```
$ cat Gemfile
source 'https://rubygems.org'
gemspec

$ cat Gemfile.lock
cat: Gemfile.lock: Aucun fichier ou dossier de ce type

$ bundle install
Fetching gem metadata from https://rubygems.org/.........
Resolving dependencies...
Using rake 11.1.1
Using ffi 1.9.10
...
Using bundler 1.7.4
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem is inst
```

Note : `bundle install` examine le fichier `Gemfile`.

Ici, ce fichier fait simplement référence au fichier `gemspec` courant.

L'outil `bundler` permet d'installer automatiquement des *gems* à partir d'un `Gemfile`

Les gems requis/utilisés sont enregistrés dans le fichier `Gemfile.lock`

```
$ cat Gemfile.lock
PATH
  remote: .
  specs:
    foo (0.0.1)
      gli (= 2.12.0)

GEM
  remote: https://rubygems.org/
  specs:
    aruba (0.14.1)
      childprocess (~> 0.5.6)
      contracts (~> 0.9)
      cucumber (>= 1.3.19)
      ffi (~> 1.9.10)
      rspec-expectations (>= 2.99)
    ...
    rdoc (4.2.2)
      json (~> 1.4)
    rspec-expectations (3.4.0)
      diff-lcs (>= 1.2.0, < 2.
      rspec-support (~> 3.4.0)
    rspec-support (3.4.1)
    thor (0.19.1)

PLATFORMS
  ruby

DEPENDENCIES
  aruba
  rake
  rdoc
```

Note : Si on développe une application — et non un *gem* — alors le fichier `Gemfile.lock` devrait être mis sous contrôle des versions.

Remarque importante concernant `gem install` et `bundle install`

- Dans certaines configurations, par défaut, ces commandes installent les *gems* dans un répertoire global à tous les usagers

  $\Rightarrow$ Il faut être `root` ou exécuter en `sudo` ☹

Remarque importante concernant `gem install` et `bundle install`

- Dans certaines configurations, par défaut, ces commandes installent les *gems* dans un répertoire global à tous les usagers

  $\Rightarrow$ Il faut être `root` ou exécuter en `sudo` ☹

- Par exemple, sur `malt`, vous ne pouvez pas installer de *gems* de cette façon !

Remarque importante concernant `gem install` et `bundle install`

- Dans certaines configurations, par défaut, ces commandes installent les *gems* dans un répertoire global à tous les usagers

  $\Rightarrow$ Il faut être `root` ou exécuter en `sudo` ☹

- Par exemple, sur `malt`, vous ne pouvez pas installer de *gems* de cette façon !

- Par contre, si nécessaire, vous pouvez installer des *gems* dans votre espace personnel :

```
$ bundle install --path vendor/bundle
```

3.5 En résumé

En résumé, pour un projet non trivial `foo` comportant des dépendances envers de nombreux *gems*

1. On définit un fichier `foo.gemspec` avec les dépendances (i.e., les *gems*) requises

En résumé, pour un projet non trivial `foo` comportant des dépendances envers de nombreux *gems*

1. On définit un fichier `foo.gemspec` avec les dépendances (i.e., les *gems*) requises

2. On définit un fichier `Gemfile` contenant ce qui suit :

```
source 'https://rubygems.org'
gemspec
```

En résumé, pour un projet non trivial `foo` comportant des dépendances envers de nombreux *gems*

1. On définit un fichier `foo.gemspec` avec les dépendances (i.e., les *gems*) requises

2. On définit un fichier `Gemfile` contenant ce qui suit :

```
source 'https://rubygems.org'
gemspec
```

3. On exécute :
   - `$ bundle install`
     si on peut installer soi-même des *gems* de façon globale...
   - `$ bundle install --path vendor/bundle`
     si on doit installer des *gems* dans son espace personnel

4. Le *gem* `rake` pour l'automatisation des tâches

Qu'est-ce que `rake` ?

Rake is a build language, similar in purpose to make and ant. Like make and ant it's a Domain Specific Language, unlike those two it's an internal DSL programmed in the Ruby language.

Source*: http ://martinfowler.com/articles/rake.html*

Qu'est-ce que `rake` ?

Rake is a build language, similar in purpose to make and ant. Like make and ant it's a Domain Specific Language, unlike those two it's an internal DSL programmed in the Ruby language.

Source: *http ://martinfowler.com/articles/rake.html*

[Rake] came about as an experiment to see if Make's functionality could be easily reproduced by creating an internal DSL in Ruby. The answer was "yes," and Rake was born.

Source: *«Continuous Delivery», Humble & Farley*

Caractéristiques de `rake`

- Comme `Make` : `rake` se fonde uniquement sur les notions de tâches et de dépendances.
- Comme `Make` : `rake`, bien qu'écrit en Ruby, peut être utilisé avec n'importe quel langage
- Contrairement à `Make` : un script `rake` est un programme Ruby comme n'importe quel autre (DSL interne) !
- Contrairement à `Make` : les commandes (e.g., manipulation de fichiers) d'un script `rake` peuvent être écrites de façon indépendante de la plateforme

Exemple = compilation d'un fichier `.c` : Avec `Make`

```
default: run

# Execution du programme.
run: hello
	./hello

# Generation de l'executable.
hello: hello.c
	gcc -o hello hello.c

# Nettoyage.
clean:
	rm -f hello hello.o
```

Exemple = compilation d'un fichier `.c` : Avec `rake`

```
task :default => :run

desc 'Execution du programme'
task :run => 'hello' do
  sh %{./hello}
end

desc 'Generation de l\'executable'
file 'hello' => ['hello.c'] do
  sh %{gcc -o hello hello.c}
end

desc 'Nettoyage'
task :clean do
  rm_f 'hello hello.o'
end
```

Quelques exemples d'exécution

Construction de la cible par défaut

```
$ rake
gcc -o hello hello.c
./hello
Hello, World!
```

Quelques exemples d'exécution

Construction de la cible par défaut

```
$ rake
gcc -o hello hello.c
./hello
Hello, World!
```

Construction d'une cible déjà dérivée ⇒ aucune action

```
$ rm hello

$ rake hello
gcc -o hello hello.c

$ rake hello

$
```

Quelques exemples d'exécution

Liste des tâches disponibles, avec description

```
$ rake -T
rake clean  # Nettoyage
rake hello  # Generation de l'executable
rake run    # Execution du programme
```

Liste des tâches, avec ou sans description

```
$ rake -T -A
rake clean    # Nettoyage
rake default  #
rake hello    # Generation de l'executable
rake run      # Execution du programme
```

Quelques exemples d'exécution

Dry run — indique ce qui serait exécuté, mais sans exécuter

```
$ rake -n
** Invoke default (first_time)
** Invoke run (first_time)
** Invoke hello (first_time, not_needed)
** Invoke hello.c (first_time, not_needed)
** Execute (dry run) run
** Execute (dry run) default
```

L'utilisation de règles implicites permet de réduire le nombre de tâches explicitement spécifiées ⋆

Règle style `Make`

```
rule '.o' => '.c' do |task|
  sh %{gcc -o #{task.name} #{task.source}}
end
```

L'utilisation de règles implicites permet de réduire le nombre de tâches explicitement spécifiées ⋆

Règle avec *pattern-matching*

```
rule /\.o$/ => '.c' do |task|
  sh %{gcc -o #{task.name} #{task.source}}
end
```

L'utilisation de règles implicites permet de réduire le nombre de tâches explicitement spécifiées ⋆

Règle avec détection dynamique

```
articles = Rake::FileList.new( '**/*.md' ) do |files|
  files.exclude( '~*' )
  files.exclude( /^temp.+\// )
  files.exclude { |file| File.zero? file }
  ...
end

detect_files = lambda do |task|
  articles.detect { |article| article.ext == task.ext }
end

rule '.html' => detect_file do |task|
   sh %{...}
end
```

Source: Adapté/simplifié de «*Rake Task Management Essentials*», Koleshko

Les avantages/désavantages de `rake`

Avantages

- DSL Interne $\Rightarrow$ Puissant et expressif ☺
- Disponible sur toute plateforme où Ruby est disponible

  Notamment, `jruby` fonctionne sur toute plateforme avec une JVM (machine virtuelle Java)

Les avantages/désavantages de `rake`

Avantages

- DSL Interne ⇒ Puissant et expressif ☺
- Disponible sur toute plateforme où Ruby est disponible

  Notamment, `jruby` fonctionne sur toute plateforme avec une JVM (machine virtuelle Java)

Désavantages

- DSL Interne ⇒ Il faut connaitre (un peu) Ruby ☹
- Il faut installer Ruby et divers *gems*
- `jruby` est parfois un peu lent à démarrer ⇒ un appel à `rake` avec `jruby` est plus lent qu'un appel à `make`

5. Le *gem* `gli` pour la spécification de suites de commandes

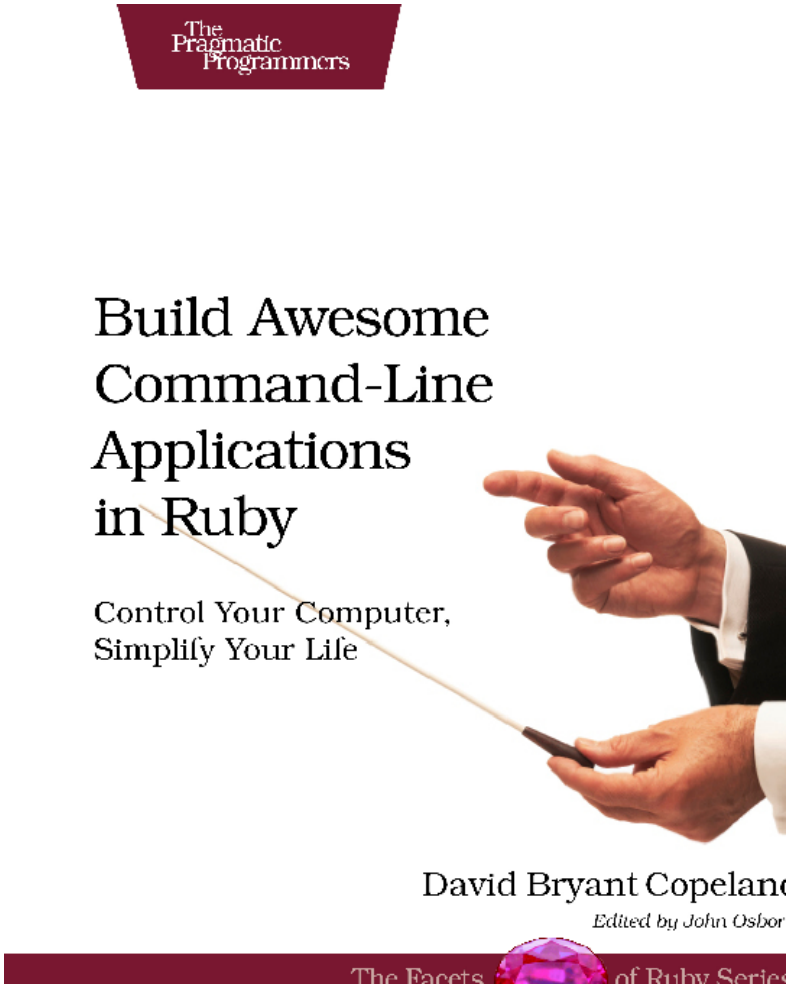
Le *gem* `gli` est utilisé pour spécificier des suites de commandes «à la `git`»

GLI lets you create command-suite (i.e. git-like) style applications in Ruby very easily.

Source: `https://github.com/davetron5000/gli/wiki`

`gli` est décrit dans le livre de Copeland (2012)

`gli` = *git like interface*
command line parser



Le *gem* `gli` fournit un DSL pour la spécification de suite de commandes

DSL = *Domain-Specific Language*

A [programming] language offering expressive power focused on a particular problem domain, such as a specific class of applications or application aspect.

Source: *K. Czarnecki, 2005*

Parfois aussi appelé un «petit langage» — «*a little language*».

Note : On traitera des DSLs dans un prochain cours.

Le *gem* `gli` définit diverses méthodes qui visent à définir les options et les commandes d'une application

Description d'options

```
desc "Description d'une switch"
switch [:o, :option]

desc "Description d'un flag"
default_value "Valeur par defaut"
arg_name "Nom de l'argument"
flag [:f, :flag]
```

Le *gem* `gli` définit diverses méthodes qui visent à définir les options et les commandes d'une application

Description d'options

```
desc "Description d'une switch"
switch [:o, :option]

desc "Description d'un flag"
default_value "Valeur par defaut"
arg_name "Nom de l'argument"
flag [:f, :flag]
```

Éléments du DSL = méthodes fournies par `gli`

- `desc`
- `switch`
- `default_value`
- `arg_name`
- `flag`

Le *gem* `gli` définit diverses méthodes qui visent à définir les options et commandes d'une application

Description de commandes

```
desc 'Description de la commande'
arg_name 'Liste des arguments'
command :nom_de_la_commande do |c|
  c.desc 'Option pour commande'
  c.switch :s

  c.action do |global_options,options,args|
    # Mise en oeuvre de la commande
    ...
  end
end
```

Le *gem* `gli` définit diverses méthodes qui visent à définir les options et commandes d'une application

Description de commandes

```
desc 'Description de la commande'
arg_name 'Liste des arguments'
command :nom_de_la_commande do |c|
  c.desc 'Option pour commande'
  c.switch :s

  c.action do |global_options,options,args|
    # Mise en oeuvre de la commande
    ...
  end
end
```

Autres éléments du DSL = méthodes fournies par `gli`

- `command`
- `action`

Le *gem* `gli` fournit aussi une commande qui permet de créer le squelette d'une application

The simplest way to get started is to create a scaffold project. For example, a command-suite app named "todo" that has the commands "list", "add" and "complete" is created with:

```
> gli init todo list add complete
```

A new `./todo` directory is created containing the app. View the basic output of the scaffold with:

```
> cd todo
> bundle exec bin/todo help
```

Source: `https://github.com/davetron5000/gli/wiki`

6. Le *gem* `mini-sed`, une version simplifiée de `sed`

6.1 La commande `sed`

La commande `sed`

`sed` = *stream editor* = éditeur de flux pour le filtrage et la transformation de texte

`man sed`

SYNOPSIS

```
sed [options]... {script-seulement-si-pas-d-autre-script
                 [fichier-d-entrée]...
```

DESCRIPTION

`sed` est un éditeur de flux. Un éditeur de flux est utilisé pour effectuer des transformations de texte basiques sur un flux d'entrée (un fichier ou l'entrée d'un tube). Alors que d'une certaine manière il est similaire à un éditeur qui permet des éditions scriptées (comme `ed`), `sed` fonctionne en seulement une passe sur l'entrée(s) et est, par conséquent, plus efficace. Mais c'est sa capacité à filtrer du texte dans un tube qui le distingue des autres éditeurs.

Note : Dans ce qui suit, on examine quelques-unes des (très !) nombreuses expressions d'édition et options.

La commande `sed`

Quelques expressions (scripts) d'édition... qui s'exécutent sur chaque ligne du flux d'entrée

`/patron/d`	Supprime la ligne si elle matche *patron*
`/patron/p`	Imprime la ligne si elle matche *patron*
`s/patron/chaine/`	Substitue la première occurrence de *patron* par *chaine*
`s/patron/chaine/g`	Substitue toutes les occurrences de *patron* par *chaine*

Notes :

- Chaque expression d'édition s'applique sur chacune des lignes du flux d'entrée et la ligne modifiée est émise en sortie, sauf si l'action est `d`.
- Le caractère utilisé pour délimiter la commande, le patron et la chaine peut être n'importe quel caractère répété (pas juste «/»)

La commande `sed`

Une option

Option	**Signification**
`--in-place[=EXT]`	Édite le fichier en ligne Fait une sauvegarde si une `EXT`ension est spécifiée

La commande sed

Quelques exemples de suppression

```
$ cat fich.txt
1 2 3
xxx yyy zzz
d/e/f
```

Suppression

```
$ sed '/xx/d' fich.txt
1 2 3
d/e/f
```

La commande `sed`

Quelques exemples avec impression explicite

```
$ cat fich.txt
1 2 3
xxx yyy zzz
d/e/f
```

Impression explicite

```
$ sed '/xx/p' fich.txt
1 2 3
xxx yyy zzz
xxx yyy zzz
d/e/f
```

La commande sed

Quelques exemples de substitution

```
$ cat fich.txt
1 2 3
xxx yyy zzz
d/e/f
```

Substitutions simples

```
$ sed 's/x/ABC/' fich.txt
1 2 3
ABCxx yyy zzz
d/e/f

$ sed 's/x/ABC/g' fich.txt
1 2 3
ABCABCABC yyy zzz
d/e/f

$ sed 's/\(.\)\1\1/\1/g' fich.txt
??
```

La commande sed

Quelques exemples de substitution

```
$ cat fich.txt
1 2 3
xxx yyy zzz
d/e/f
```

Substitutions simples

```
$ sed 's/x/ABC/' fich.txt
1 2 3
ABCxx yyy zzz
d/e/f

$ sed 's/x/ABC/g' fich.txt
1 2 3
ABCABCABC yyy zzz
d/e/f

$ sed 's/\(.\)\1\1/\1/g' fich.txt
1 2 3
x y z
d/e/f
```

La commande `sed`

Pseudocode simplifié (! !), pour comprendre le fonctionnement de base de `sed`

```
while !STDIN.eof? do
   ligne ← STDIN.readline
   emettre_ligne ← true  # On l'emet... sauf si action = 'd'

   if ligne matche le motif then
     case action
       when 'd':
         emettre_ligne ← false
       when 'p':
         STDOUT.print ligne
       when 's':
         modifier ligne selon la substitution indiquée
     end
   end

   STDOUT.print ligne if emettre_ligne
end
```

6.2 Le *gem* `mini-sed`

L'exemple de *gem* qui suit est une version simplifiée de `sed`, avec des commandes et options «à la `git`»

- Présente les principales étapes pour la création d'une application avec une interface personne–machine en ligne de commandes.
- Illustre aussi le processus de développement d'une petite application — dont certain-e-s erreurs/problèmes rencontré-e-s.
- Mise en oeuvre = L'analyse et la répartition (*dispatch*) des commandes et options se fait en utilisant le *gem* `gli`.

Voici à quoi ressemblera l'interface de notre version simplifiée de `sed` «à la `git`»

```
# sed '/m1/d' ...
$ bin/mini-sed delete m1 ...


# sed 's/m1/c1/' ...
$ bin/mini-sed substitute m1 c1 ...


# sed 's/m1/c1/g' ...
$ bin/mini-sed substitute -g m1 c1 ...


# sed --in-place=.bak 's/m1/c1/g' ...
$ bin/mini-sed --in_place=.bak substitute -g m1 c1 ...
```

6.3 Utilisation de `gli` pour créer le squelette de `mini-sed`

On doit tout d'abord installer le *gem* `gli`

```
$ gem install gli
Fetching: gli-2.13.2.gem (100%)
Successfully installed gli-2.13.2
1 gem installed
```

Remarque importante : Si vous travaillez sur `malt`, vous n'avez pas besoin d'installer `gli` car il est déjà disponible — vous n'auriez pas les permissions pour l'installer de toute façon ☹

On crée le squelette de l'application `mini-sed`

Les arguments après le nom de l'application (`mini-sed`) sont les commandes que l'application doit avoir : `substitute`, `delete` et `print`

```
$ gli scaffold mini-sed substitute delete print
Creating dir ./mini-sed/lib...
Creating dir ./mini-sed/bin...
Creating dir ./mini-sed/test...
Created ./mini-sed/bin/mini-sed
Created ./mini-sed/README.rdoc
Created ./mini-sed/mini-sed.rdoc
Created ./mini-sed/mini-sed.gemspec
Created ./mini-sed/test/default_test.rb
Created ./mini-sed/test/test_helper.rb
Created ./mini-sed/Rakefile
Created ./mini-sed/Gemfile
Created ./mini-sed/features
Created ./mini-sed/lib/mini-sed/version.rb
Created ./mini-sed/lib/mini-sed.rb
```

Voici alors la structure des répertoires pour le squelette d'application créé par `gli`

```
$ tree mini-sed
mini-sed
|-- bin
|   `-- mini-sed
|-- features
|
|-- mini-sed.feature
|
|-- step_definitions
|   |
`-- mini-sed_steps.rb
|   `-- support
|       `-- env.rb
|-- Gemfile
|-- lib
|   |-- mini-sed
|   |   `-- version.rb
|   `-- mini-sed.rb
|-- mini-sed.gemspec
|-- mini-sed.rdoc
|-- Rakefile
|-- README.rdoc
`-- test
    |-- default_test.rb
    `-- test_helper.rb

7 directories, 13 files
```

Contenu du programme principal : Chargement des *gems* requis et inclusion de `GLI::App`

```
$ cat mini-sed/bin/mini-sed
#!/usr/bin/env ruby
require 'gli'
begin # XXX: Remove this begin/rescue before distributin
require 'mini-sed'
rescue LoadError
  STDERR.puts "In development, you need to use `bundle e
  STDERR.puts "At install-time, RubyGems will make sure
  STDERR.puts "Feel free to remove this message from bin
  exit 64
end

include GLI::App

program_desc 'Describe your application here'

version MiniSed::VERSION
```

Contenu du programme principal (suite) : Spécification des options globales

```
# Use argument validation
arguments :strict

desc 'Describe some switch here'
switch [:s,:switch]

desc 'Describe some flag here'
default_value 'the default'
arg_name 'The name of the argument'
flag [:f,:flagname]
```

Contenu du programme principal (suite) : Squelette de mise en oeuvre de la commande `substitute`

```
desc 'Describe substitute here'
arg_name 'Describe arguments to substitute here'
command :substitute do |c|
  c.desc 'Describe a switch to substitute'
  c.switch :s

  c.desc 'Describe a flag to substitute'
  c.default_value 'default'
  c.flag :f
  c.action do |global_options,options,args|

    # Your command logic here
    # If you have any errors, just raise them
    # raise "that command made no sense"

    puts "substitute command ran"
  end
end
```

Contenu du programme principal (suite) : Squelettes de mise en oeuvre des commandes `delete` et `print`

```
desc 'Describe delete here'
arg_name 'Describe arguments to delete here'
command :delete do |c|
  c.action do |global_options,options,args|
    puts "delete command ran"
  end
end

desc 'Describe print here'
arg_name 'Describe arguments to print here'
command :print do |c|
  c.action do |global_options,options,args|
    puts "print command ran"
  end
end
```

Contenu du programme principal (suite) : Traitement des erreurs et lancement de l'exécution

```
...

on_error do |exception|
  # Error logic here
  # return false to skip default error handling
  true
end

exit run(ARGV)
```

On met les fichiers sous contrôle de `git`

```
$ cd mini-sed
```

```
$ git init .
Initialized empty Git repository in /home/tremblay/mini-sed/.g
```

```
$ git add .
```

```
$ git commit -am "Creation initiale du projet"
[master (root-commit) 81c9f08] Creation initiale
 13 files changed, 225 insertions(+)
 create mode 100644 Gemfile
 create mode 100644 README.rdoc
 create mode 100644 Rakefile
 create mode 100755 bin/mini-sed
 ...
 create mode 100644 test/default_test.rb
 create mode 100644 test/test_helper.rb
```

On installe les *gems* requis avec `bundle`

```
$ bundle install
Fetching gem metadata from https://rubygems.org/.......
Resolving dependencies...
Using rake 11.1.1
Using ffi 1.9.10
...
Using json 1.8.3
Using mini-sed 0.0.1 from source at .
Using rdoc 4.2.2
Using bundler 1.7.4
Your bundle is complete!
Use `bundle show [gemname]` to see where a bundled gem i
```

Remarque importante

Sur `malt`, vous devez plutôt exécuter la commande suivante :

```
$ bundle install --path vendor/bundle
```

Un exemple d'exécution directe du programme squelette

Les *gems* installés pour ce projet ne sont pas ceux disponibles par défaut ☹

```
$ bin/mini-sed
In development, you need to use
   `bundle exec bin/mini-sed` to run your app
At install-time, RubyGems will make sure lib, etc.
   are in the load path
Feel free to remove this message from bin/mini-sed now
```

Un exemple d'exécution indirecte du programme squelette : Aucun argument ⇒ `help`

`bunble exec` assure que les *gems* utilisés seront ceux pour le projet

```
$ bundle exec bin/mini-sed
```

```
NAME
    mini-sed - Describe your application here
SYNOPSIS
    mini-sed [global options] command [command options] [argume
VERSION
    0.0.1
GLOBAL OPTIONS
    -f, --flagname=The name of the argument - Describe some fl
    --help                                  - Show this messag
    -s, --[no-]switch                       - Describe some sw
    --version                               - Display the prog
COMMANDS
    delete      - Describe delete here
    help        - Shows a list of commands or help for one comm
    print       - Describe print here
    substitute - Describe substitute here
```

Un autre exemple d'exécution : la commande `help substitute`

Affiche un squelette d'aide... évidemment à compléter

```
$ bundle exec bin/mini-sed help substitute
NAME
    substitute - Describe substitute here

SYNOPSIS
    mini-sed [global options] substitute
             [command options] Describe arguments to sub

COMMAND OPTIONS
    -f arg - Describe a flag to substitute (default: def
    -s     - Describe a switch to substitute
```

Un autre exemple d'exécution : la commande `substitute`

Exécution de la commande bidon, qui affiche simplement une trace

```
$ bundle exec bin/mini-sed substitute
substitute command ran
```

Par défaut, les tests unitaires utilisent `Test::Unit`

Par contre, dans la suite de cet exemple, on va plutôt utiliser `MiniTest`

```
$ cat test/test_helper.rb
require 'test/unit'
# Add test libraries you want to use here, e.g. mocha

class Test::Unit::TestCase
  # Add global extensions to the test case class here
end
```

```
$ cat test/default_test.rb
require 'test_helper'

class DefaultTest < Test::Unit::TestCase
  def setup
  end
  ...
  def test_the_truth
    assert true
  end
end
```

Voici le contenu de ces deux fichiers après modification pour plutôt utiliser `MiniTest`

```
$ cat test/test_helper.rb
gem 'minitest', '=5.4.3'
require 'minitest/autorun'
require 'minitest/spec'
require 'minitest/mock'

class Object
  def _it( test ) ... end
end

$ cat test/default_test.rb
require 'test_helper'
require 'mini-sed'

describe "A COMPLETER" do
  it "a completer" do
  end
end
```

Par défaut, les tests d'acception sont configurés pour être décrits avec `cucumber` et `aruba`

On y reviendra ultérieurement...

```
$ cat mini-sed/features/mini-sed.feature
Feature: My bootstrapped app kinda works
  In order to get going on coding my awesome app
  I want to have aruba and cucumber setup
  So I don't have to do it myself

  Scenario: App just runs
    When I get help for "mini-sed"
    Then the exit status should be 0
```

```
$ cat mini-sed/features/step_definitions/mini-sed_steps.
When /^I get help for "([^"]*)"$/ do |app_name|
  @app_name = app_name
  step %(I run `#app_name help`)
end

# Add more step definitions here
```

6.4 Mise en oeuvre simple de la commande `substitute` et exemples d'exécution

Brève description de la suite de la présentation ⋆

- L'approche BDD/TDD suggère de commencer en définissant les tests.
- Mais, ce n'est pas toujours facile de procéder ainsi quand on ne connaît pas un *framework*, un outil, un langage.
- Pour illustrer les fonctionnalités de `gli`, nous allons débuter la mise en oeuvre du programme principal en spécifiant les options globales et la commande `substitute`.
  Le programme va simplement analyser les options et arguments, puis afficher les informations fournies.
- Donc, pour l'instant, il n'y a pas de test. . . juste des petits exemples d'exécution.

Le programme principal `bin/mini-sed`

Spécification des options globales

```
$ cat bin/mini-sed
...
program_desc 'Programme qui emule sed -- de facon (tres)

version MiniSed::VERSION

# Use argument validation
arguments :strict

desc 'Execution avec modification directe du fichier tra
default_value ''
arg_name 'extension'
flag [:i,:in_place]
...
```

Le programme principal `bin/mini-sed`

Mise en oeuvre bidon de `substitute` : on affiche les options et arguments

```
desc "Substitue un motif par une chaine dans les lignes"
arg_name 'motif chaine [fichier...]'
command :substitute do |c|
  c.desc 'Substitution globale'
  c.switch :g

  c.action do |global_options,options,args|
    in_place = "--in-place='#{global_options[:in_place]}'"
               if global_options[:in_place]
    global = 'g' if options[:g]
    motif = args.shift
    chaine = args.shift

    # On affiche la commande sed equivalente (temporaire
    puts "sed #{in_place}" <<
              "'s/#{motif}/#{chaine}/#{global}'" <<
              " #{args.join(' ')}"
  end
end
```

Exécution de la commande `help` après les modifications

```
$ bundle exec bin/mini-sed help
NAME
    mini-sed - Programme qui emule sed -- de facon (tres) simpl
SYNOPSIS
    mini-sed [global options] command [command options] [argume
VERSION
    0.0.1
GLOBAL OPTIONS
    --help                          - Show this message
    -i, --in_place=extension - Execution avec modification dir
    --version                       - Display the program version
COMMANDS
    delete     - Describe delete here
    help       - Shows a list of commands or help for one comma
    print      - Describe print here
    substitute - Substitue un motif par une chaine dans les li
```

On ajoute, dans le `Rakefile`, une tâche pour lancer l'exécution d'appels simples

```
$ cat Rakefile
...
MINI_SED = 'bundle exec bin/mini-sed'
...

task :exemples_diapos do
  sh MINI_SED + ' substitute "[a-z]*.[0-9]" XXX'
  sh MINI_SED + ' --in_place=bak substitute -g "foo" "ba
end

...
```

Exécution de la tâche avec les exemples simples pour `substitute`

```
$ rake exemples_diapos
bundle exec bin/mini-sed substitute "[a-z]*.[0-9]" XXX
sed 's/[a-z]*.[0-9]/XXX/'

bundle exec bin/mini-sed --in_place='bak'\
       substitute -g "foo" "bar" Rakefile README.doc
sed --in-place=bak 's/foo/bar/g' Rakefile README.doc
```

6.5 Spécification et mise en oeuvre de `Minised.substitute`

Prochaine étape...

J'ai ajouté ou modifié divers fichiers :

- `lib/dbc.rb` : Méthodes pour DBC (*Design By Contract*)
- `lib/debug.rb` : Méthodes pour aider au débogage (trace dans un fichier de journalisation)

- `lib/mini-sed.rb` : Contient les `require` : Voir plus loin.
- `lib/mini-sed/mini-sed.rb` : Contient diverses méthodes auxiliaires, dans un `module MiniSed`.

On regarde maintenant le nouvel état de ces fichiers et, surtout, leur structure...

Organisation des fichiers dans le répertoire `lib`

Le répertoire `lib` contient tous les fichiers associés à la mise en oeuvre, à l'exception du programme principal

```
$ tree bin
bin
`-- mini-sed

0 directories, 1 file
```

```
$ tree lib
lib
|-- dbc.rb
|-- debug.rb
|-- mini-sed
|   |-- mini-sed.rb
|   `-- version.rb
`-- mini-sed.rb

1 directory, 5 files
```

Le fichier `lib/mini-sed.rb` avec les `require`

```
$ cat lib/mini-sed.rb
# Add requires for other files you add to your project h
# you just need to require this one file in your bin fil

require 'debug'
require 'dbc'

require 'mini-sed/version'
require 'mini-sed/mini-sed'
```

- Inclut (avec `require`) tous les fichiers (modules et classes) définis par l'application.
- Donc : regroupe tous les `require`, pour éviter d'en avoir à plusieurs endroits, pour mieux identifier les fichiers/*gems* utilisés. . .

Le fichier `lib/mini-sed/version.rb`

Définit le numéro de version du *gem*

```
$ cat lib/mini-sed/version.rb
module MiniSed
  VERSION = '0.0.1'
end
```

Le fichier `lib/mini-sed/mini-sed.rb`

Définit les méthodes auxiliaires, utilisées notamment par le programme principal

```
$ cat lib/mini-sed/mini-sed.rb
module MiniSed
  def self.substitute( motif, remplacement,
                       global, lignes )
    ...
  end
end
```

Voir plus loin pour les détails de mise en oeuvre.

Mise en oeuvre révisée de `bin/mini-sed` — en ignorant l'option `--in_place`

```
command :substitute do |c|
  c.desc 'Substitution globale'
  c.switch :g

  c.action do |global_options,options,args|
    in_place = "--in-place='#{global_options[:in_place]}' " if
    global = 'g' if options[:g]

    motif = args.shift
    chaine = args.shift

    fichiers = args.empty? ? [IO] : args.map{|f| File.open(f)}
    fichiers.each do |f|
      puts MiniSed.substitute(motif, chaine, global, f.readlines)
    end
  end
end
```

Le fichier `test/substitute_test.rb` (extraits)

On teste le comportement au niveau des lignes reçues vs. émises, indépedamment de toute notion de fichiers

```
describe ".substitute" do
  let(:nb_lignes) { 100 }
  let(:lignes) { (1..nb_lignes).map { "abc\n" } }

  it "retourne [] lorsque recoit []" do
    MiniSed.substitute( nil, nil, false, [] )
      .must_be_empty
  end

  it "ne change rien lorsque le motif ne matche pas" do
    MiniSed.substitute( "X", "x", true, lignes )
      .must_equal lignes
  end
  ...
```

Le fichier `test/substitute_test.rb` (extraits) (suite)

```
describe "substitution globale ou non" do
  let(:juste_des_a) { ["aaadef\n", "123aaa\n"] }

  it "effectue le remplacement a un seul endroit
      lorsque global est false" do
    MiniSed.substitute( "a", "X", false, juste_des_a )
     .must_equal ["Xaadef\n", "123Xaa\n"]
  end

  it "effectue le remplacement a plusieurs endroits
      lorsque global est true" do
    MiniSed.substitute( "a", "X", true, juste_des_a )
     .must_equal ["XXXdef\n", "123XXX\n"]
  end
end
...
```

Le fichier `lib/mini-sed/mini-sed.rb`, commande `substitute`

Méthodes de classe de `MiniSed`, qui mettent en oeuvre les opérations clés

```
$ cat lib/mini-sed/mini-sed.rb
module MiniSed
  def self.substitute( motif, remplacement,
                       global, lignes )
    lignes.select do |ligne|
      if /#{motif}/ =~ ligne
        ligne.send (global ? :gsub! : :sub!),
                   /#{motif}/, remplacement
      end
    end
  end
end
```

Note : `sub!` et `gsub!` sont définies dans `String` :

`https://ruby-doc.org/core-2.2.0/String.html`

On lance les tests

```
$ rake test TEST=test/substitute_test.rb
Run options: --seed 33989

# Running:

......

Finished in 0.001335s, 4493.8367 runs/s, 5242.8095 asser

6 runs, 7 assertions, 0 failures, 0 errors, 0 skips
```

On a quelque chose qui fonctionne… donc, on fait un *commit* !

```
$ git commit -am "Premiere mise en oeuvre de substitute"
[master 22bf120] Premiere mise en oeuvre de substitute
 ...
```

6.6 Spécification de tests d’acceptation

Question : On veut aussi des tests du programme, au niveau de la ligne de commandes. Comment faire ?

Question : On veut aussi des tests du programme, au niveau de la ligne de commandes. Comment faire ?

Diverses solutions possibles :

1. Script/s *bash*

Question : On veut aussi des tests du programme, au niveau de la ligne de commandes. Comment faire ?

Diverses solutions possibles :

1. Script/s *bash*

2. Cadre/outil de tests d'acceptation — par ex., `cucumber`

Question : On veut aussi des tests du programme, au niveau de la ligne de commandes. Comment faire ?

Diverses solutions possibles :

1. Script/s *bash*

2. Cadre/outil de tests d'acceptation — par ex., `cucumber`
   ... dans un prochain cours

Question : On veut aussi des tests du programme, au niveau de la ligne de commandes. Comment faire ?

Diverses solutions possibles :

1. Script/s *bash*

2. Cadre/outil de tests d'acceptation — par ex., `cucumber` . . . dans un prochain cours

3. `MiniTest` + possibilité de lancer l'exécution de programmes externes = *glue language*

Un 1er test d'acceptation pour `substitute`

Représente le contenu d'un fichier par une longue chaine...

```
it "change seulement la 1ere occurrence sans -g" do
  # Setup.
  nom_fichier = 'foo.txt'
  contenu = "0abc!\n==\nabcdef9abc\n" # Longue chaine!
  File.open( nom_fichier, "w" ) do |fich|
    fich.print contenu
  end

  # Exercise
  cmd = 'substitute "abc" "XX" <foo.txt'
  obtenu = %x{bundle exec bin/mini-sed #{cmd}}

  # Verify.
  obtenu.must_equal "0XX!\n==\nXXdef9abc\n" # Longue chaine!

  # Tear down.
  FileUtils.rm_f nom_fichier
end
```

Un 1er test d'acceptation pour `substitute`

Représente le contenu d'un fichier par une longue chaine... difficile à lire ☹

```
it "change seulement la 1ere occurrence sans -g" do
  # Setup.
  nom_fichier = 'foo.txt'
  contenu = "0abc!\n==\nabcdef9abc\n" # Longue chaine!
  File.open( nom_fichier, "w" ) do |fich|
    fich.print contenu
  end

  # Exercise
  cmd = 'substitute "abc" "XX" <foo.txt'
  obtenu = %x{bundle exec bin/mini-sed #{cmd}}

  # Verify.
  obtenu.must_equal "0XX!\n==\nXXdef9abc\n" # Longue chaine!

  # Tear down.
  FileUtils.rm_f nom_fichier
end
```

Un 1er test d'acceptation pour `substitute`

Représente le contenu d'un fichier par une liste de chaines, sans «\n»

```
it "change seulement la 1ere occurrence sans -g" do
  # Setup.
  nom_fichier = 'foo.txt'
  contenu = ["0abc!", "==", "abcdef9abc"] # Liste de lignes!
  File.open( nom_fichier, "w" ) do |fich|
    contenu.each { |ligne| fich.puts ligne }
  end

  # Exercise
  cmd = 'substitute "abc" "XX" <foo.txt'
  obtenu = %x{bundle exec bin/mini-sed #{cmd}}.split("\n")

  # Verify.
  obtenu.must_equal ["0XX!", "==", "XXdef9abc"] # Liste de lig

  # Tear down.
  FileUtils.rm_f nom_fichier
end
```

Un 2^{e} test d'acceptation pour `substitute`

Quel est le problème ?

```
it "modifie chaque ligne qui matche avec plusieurs fichiers" d
  # Setup.
  nom_fichier = 'foo.txt'
  contenu = ["aaax2xaax2", "==", "abcabc32zz22"]
  File.open( nom_fichier, "w" ) do |fich|
    contenu.each { |ligne| fich.puts  ligne }
  end

  # Exercise
  cmd = 'substitute "[abc]*.[12]" "XX" foo.txt foo.txt'
  obtenu = %x{bundle exec bin/mini-sed #{cmd}}.split("\n")

  # Verify.
  obtenu.must_equal ["XXxaax2", "==", "XXzz22",
                     "XXxaax2", "==", "XXzz22"]

  # Tear down.
  FileUtils.rm_f nom_fichier
end
```

Un 2^{e} test d'acceptation pour `substitute`

Quel est le problème ? Code répétitif ! (Pas DRY !)

```
it "modifie chaque ligne qui matche avec plusieurs fichiers" d
  # Setup.
  nom_fichier = 'foo.txt'
  contenu = ["aaax2xaax2", "==", "abcabc32zz22"]
  File.open( nom_fichier, "w" ) do |fich|
    contenu.each { |ligne| fich.puts  ligne }
  end

  # Exercise
  cmd = 'substitute "[abc]*.[12]" "XX" foo.txt foo.txt'
  obtenu = %x{bundle exec bin/mini-sed #{cmd}}.split("\n")

  # Verify.
  obtenu.must_equal ["XXxaax2", "==", "XXzz22",
                     "XXxaax2", "==", "XXzz22"]

  # Tear down.
  FileUtils.rm_f nom_fichier
end
```

Le 1er test d'acceptation pour `substitute`

Mais cette fois avec des méthodes auxiliaires !

```
it "change seulement la 1ere occurrence sans -g" do
  avec_fichier 'foo.txt', ["0abc!", "==", "abcdef9abc"] do
    mini_sed( 'substitute "abc" "XX" <foo.txt' ).
      must_equal ["0XX!", "==", "XXdef9abc"]
  end
end
```

Le 1er test d'acceptation pour `substitute`

Mais cette fois avec des méthodes auxiliaires !

```
it "change seulement la 1ere occurrence sans -g" do
  avec_fichier 'foo.txt', ["0abc!", "==", "abcdef9abc"] do
    mini_sed( 'substitute "abc" "XX" <foo.txt' ).
      must_equal ["0XX!", "==", "XXdef9abc"]
  end
end
```

La méthode `avec_fichier`

```
def avec_fichier( nom_fichier, contenu )
  # Creation du fichier temporaire.
  File.open( nom_fichier, "w" ) do |fich|
    contenu.each { |ligne| fich.puts  ligne }
  end

  yield  # Execution du bloc

  FileUtils.rm_f nom_fichier  # Suppression du fichier.
end
```

Le 1er test d'acceptation pour `substitute`

Mais cette fois avec des méthodes auxiliaires !

```
it "change seulement la 1ere occurrence sans -g" do
  avec_fichier 'foo.txt', ["0abc!", "==", "abcdef9abc"] do
    mini_sed( 'substitute "abc" "XX" <foo.txt' ).
      must_equal ["0XX!", "==", "XXdef9abc"]
  end
end
```

La méthode `mini_sed`

```
def mini_sed( cmd )
  # Execute la commande indiquee.
  # Retourne un Array des lignes obtenues,
  # sans les caracteres de fin de ligne.

  %x{bundle exec bin/mini-sed #{cmd}}.split("\n")
end
```

6.7 La mise en oeuvre de l'option `--in_place`

Un 1er test d'acceptation pour l'option `--in_place`

Nécessite d'avoir une méthode qui obtient le contenu d'un fichier !

```
it "change toutes les occurrences, modifie le fichier
    et cree une copie" do
  entree = ["0abc!", "==", "abcdef9abc"]
  cmd = '--in_place=.bak substitute -g "abc" "XX" foo.txt'
  sortie = ["0XX!", "==", "XXdef9XX"]

  avec_fichier 'foo.txt', entree do
    stdout = mini_sed( cmd )

    contenu_fichier( 'foo.txt' ).must_equal sortie
    stdout.must_equal []
    contenu_fichier( 'foo.txt.bak' ).must_equal entree
  end

  FileUtils.rm_f 'foo.txt.bak'
end
```

Un 1er test d'acceptation pour l'option `--in_place`

Nécessite d'avoir une méthode qui obtient le contenu d'un fichier !

La méthode `contenu_fichier`

```
def contenu_fichier( nom_fichier )
  IO.readlines(nom_fichier).map(&:chomp)
end
```

La méthode `String#chomp`

```
>> "abc\n".chomp
=> "abc"

>> "abc".chomp
=> "abc"
```

Un 2^{e} test d'acceptation pour l'option `--in_place`

Cas avec extension vide : ne crée pas de fichier !

```
it "change toutes les occurrences, modifie le fichier sans cre
  entree = ["0abc!", "==", "abcdef9abc"]
  cmd = '--in_place="" substitute -g "abc" "XX" foo.txt'
  sortie = ["0XX!", "==", "XXdef9XX"]
  ls_avant = %x{ls -1} # Liste des fichiers dans le repertoire

  avec_fichier 'foo.txt', entree do
    stdout = mini_sed( cmd )

    stdout.must_equal []
    contenu_fichier( 'foo.txt' ).must_equal sortie
  end

  %x{ls -1}.must_equal ls_avant # La liste n'a pas change!
end
```

Mise en oeuvre révisée de la commande `substitute`

```
command :substitute do |c|
  c.desc 'Substitution globale'
  c.switch :g

  c.action do |global_options,options,args|
    ext_in_place = global_options[:in_place]
    global = options[:g]
    motif = args.shift
    chaine = args.shift
    args = [:STDIN] if args.empty? # Cas particulier pour 0 fi

    ...
```

Mise en oeuvre révisée de la commande `substitute` (suite)

```
    args.each do |nom_fichier|
      flux = nom_fichier == :STDIN ? IO
                                   : File.open(nom_fichier)
      res = MiniSed.substitute( motif, chaine, global,
                                flux.readlines )
      if ext_in_place
        DBC.assert nom_fichier != :STDIN, "*** Impossible de t
        flux.close
        FileUtils.cp nom_fichier, # On cree la copie, si neces
             "#{nom_fichier}#{ext_in_place}" unless ext_in_pla

        # On modifie le fichier recu.
        File.open(nom_fichier, "w") { |flux| flux.puts res }
      else
        puts res
      end
    end
  end
end
```

6.8 *Refactoring* de `bin/mini-sed...` après le labo

Mise en oeuvre de la commande `delete` : Pas DRY !

DRY = *Don't Repeat Yourself !* Donc, code pas DRY = Beaucoup de code répétif ☹

```
command :delete do |c|
  c.action do |global_options,options,args|
    ext_in_place = global_options[:in_place]
    motif = args.shift
    args = [STDIN] if args.empty?

    args.each do |nom_fichier|
      flux = nom_fichier == STDIN ? STDIN : File.open(nom_fich
      res = MiniSed.delete( motif, flux.readlines )
      if ext_in_place
        DBC.assert nom_fichier != STDIN, "*** Impossible de tr
        flux.close
        FileUtils.cp nom_fichier, "#{nom_fichier}#{ext_in_plac
        File.open(nom_fichier, "w") { |flux| flux.puts res }
      else
        puts res
      end
    end
  end
end
```

Mise en oeuvre de la commande `print` : Pas DRY !

DRY = *Don't Repeat Yourself !* Donc, code pas DRY = Beaucoup de code répétif ☹

```
command :print do |c|
  c.action do |global_options,options,args|
    ext_in_place = global_options[:in_place]
    motif = args.shift
    args = [STDIN] if args.empty?

    args.each do |nom_fichier|
      flux = nom_fichier == STDIN ? STDIN : File.open(nom_fich
      res = MiniSed.print( motif, flux.readlines )
      if ext_in_place
        DBC.assert nom_fichier != STDIN, "*** Impossible de tr
        flux.close
        FileUtils.cp nom_fichier, "#{nom_fichier}#{ext_in_plac
        File.open(nom_fichier, "w") { |flux| flux.puts res }
      else
        puts res
      end
    end
  end
end
```

Une première version plus «DRY» de `delete` !

```
command :delete do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args

    MiniSed.traiter_fichiers( fichiers,
                              global_options[:in_place] )
      do |flux|
        MiniSed.delete( motif, flux.readlines )
    end
  end
end
```

Note : Première version révisée que j'ai développée et qui passait tous les tests déjà définis, sauf que. . .

Une première version plus «DRY» de `print` !

```
command :print do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args

    MiniSed.traiter_fichiers( fichiers,
                              global_options[:in_place] )
      do |flux|
        MiniSed.print( motif, flux.readlines )
    end
  end
end
```

La méthode `MiniSed.each_fichier`, ajoutée dans le fichier `lib/mini-sed/mini-sed.rb`

```ruby
def self.traiter_fichiers( noms_fichiers, ext_in_place )
  noms_fichiers = [STDIN] if noms_fichiers.empty?

  noms_fichiers.each do |nom_fichier|
    flux = nom_fichier == STDIN ? STDIN : File.open(nom_fichie
    res = yield( flux )
    if ext_in_place
      DBC.assert nom_fichier != STDIN, "*** Impossible de trai
      flux.close
      FileUtils.cp nom_fichier, "#{nom_fichier}#{ext_in_place}

      File.open(nom_fichier, "w") { |flux| flux.puts res }
    else
      puts res
    end
  end
end
```

Mais. . . est-ce que cette version est d'un bon style ?

```
def self.traiter_fichiers( noms_fichiers, ext_in_place )
  noms_fichiers = [STDIN] if noms_fichiers.empty?

  noms_fichiers.each do |nom_fichier|
    flux = nom_fichier == STDIN ? STDIN : File.open(nom_fichie
    res = yield( flux )
    if ext_in_place
      DBC.assert nom_fichier != STDIN, "*** Impossible de trai
      flux.close
      FileUtils.cp nom_fichier, "#{nom_fichier}#{ext_in_place}

      File.open(nom_fichier, "w") { |flux| flux.puts res }
    else
      puts res
    end
  end
end
```

Mais. . . est-ce que cette version est d'un bon style ?

```ruby
def self.traiter_fichiers( noms_fichiers, ext_in_place )
  noms_fichiers = [STDIN] if noms_fichiers.empty?

  noms_fichiers.each do |nom_fichier|
    flux = nom_fichier == STDIN ? STDIN : File.open(nom_fichie
    res = yield( flux )
    if ext_in_place
      DBC.assert nom_fichier != STDIN, "*** Impossible de trai
      flux.close
      FileUtils.cp nom_fichier, "#{nom_fichier}#{ext_in_place}

      File.open(nom_fichier, "w") { |flux| flux.puts res }
    else
      puts res
    end
  end
end
```

Non, pas vraiment ☹

Mais. . . est-ce que cette version est d'un bon style ?

```ruby
def self.traiter_fichiers( noms_fichiers, ext_in_place )
  noms_fichiers = [STDIN] if noms_fichiers.empty?

  noms_fichiers.each do |nom_fichier|
    flux = nom_fichier == STDIN ? STDIN : File.open(nom_fichier)
    res = yield( flux )
    if ext_in_place
      DBC.assert nom_fichier != STDIN, "*** Impossible de traiter stdin in_place"
      flux.close
      FileUtils.cp nom_fichier, "#{nom_fichier}#{ext_in_place}" unless ext_in_place.empty?

      File.open(nom_fichier, "w") { |flux| flux.puts res }
    else
      puts res
    end
  end
end
```

Non, pas vraiment ☹

- Puisqu'on a une méthode publique, on devrait pouvoir définir des tests unitaires pour cette méthode.
- Or, cette méthode va traiter plusieurs fichiers, ce qui sera plus difficile à tester.
- Une solution préférable, plus facile à tester = traiter un seul fichier à la fois !

Une autre version plus «DRY» de `delete`

```
command :delete do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args
    fichiers << STDIN if fichiers.empty?

    fichiers.each do |fichier|
      MiniSed.traiter_fichier( fichier,
                               global_options[:in_place] )
      do |flux|
        MiniSed.delete( motif, flux.readlines )
      end
    end
  end
end
```

Une autre version plus «DRY» de `print`

```
command :print do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args
    fichiers << STDIN if fichiers.empty?

    fichiers.each do |fichier|
      MiniSed.traiter_fichier( fichier,
                               global_options[:in_place] )
      do |flux|
        MiniSed.print( motif, flux.readlines )
      end
    end
  end
end
```

La méthode `MiniSed.traiter_fichier` révisée

```ruby
def self.traiter_fichier( nom_fichier, ext_in_place = nil )
  flux = nom_fichier == STDIN ? STDIN : File.open(nom_fichier)

  resultat = yield( flux )

  if ext_in_place
    DBC.assert nom_fichier != STDIN, "*** Impossible de traite
    flux.close
    FileUtils.cp nom_fichier, "#{nom_fichier}#{ext_in_place}"\
          unless ext_in_place.empty?

    # open/close explicites pour les tests unitaires!
    flux = File.open(nom_fichier, "w")
    flux.puts resultat
    flux.close
  else
    puts resultat
  end
end
```

Des tests unitaires pour `traiter_fichier`

Extraits du fichier `test/traiter_fichier_test.rb` : Traitement de `STDIN`

```
it "genere une erreur lorsqu'on traite en place" do
  -> { MiniSed.traiter_fichier(STDIN, ''){|flux|} }
          .must_raise DBC::Failure
end

it "lit stdin et emet sur stdout lorsqu'ne traite pas en
  lignes =  ["abc\n", "def\n"]

  puts_sur_stdout = []
  STDIN.stub :readlines, lignes do
    STDOUT.stub :puts, ->(l){ puts_sur_stdout << l } do
      MiniSed.traiter_fichier( STDIN ) do |flux|
        flux.readlines
      end
    end
  end

  puts_sur_stdout.must_equal [lignes]
end
```

Des tests unitaires pour `traiter_fichier`

Les tests passent avec succès, donc on fait un `commit`

```
$ make test
Run options: --seed 5726

# Running:

....................................

Finished in 0.012355s, 2913.8001 runs/s, 4694.4557 asser

36 runs, 58 assertions, 0 failures, 0 errors, 0 skips

$ git add ...
...

$ git commit -m "..."
...
```

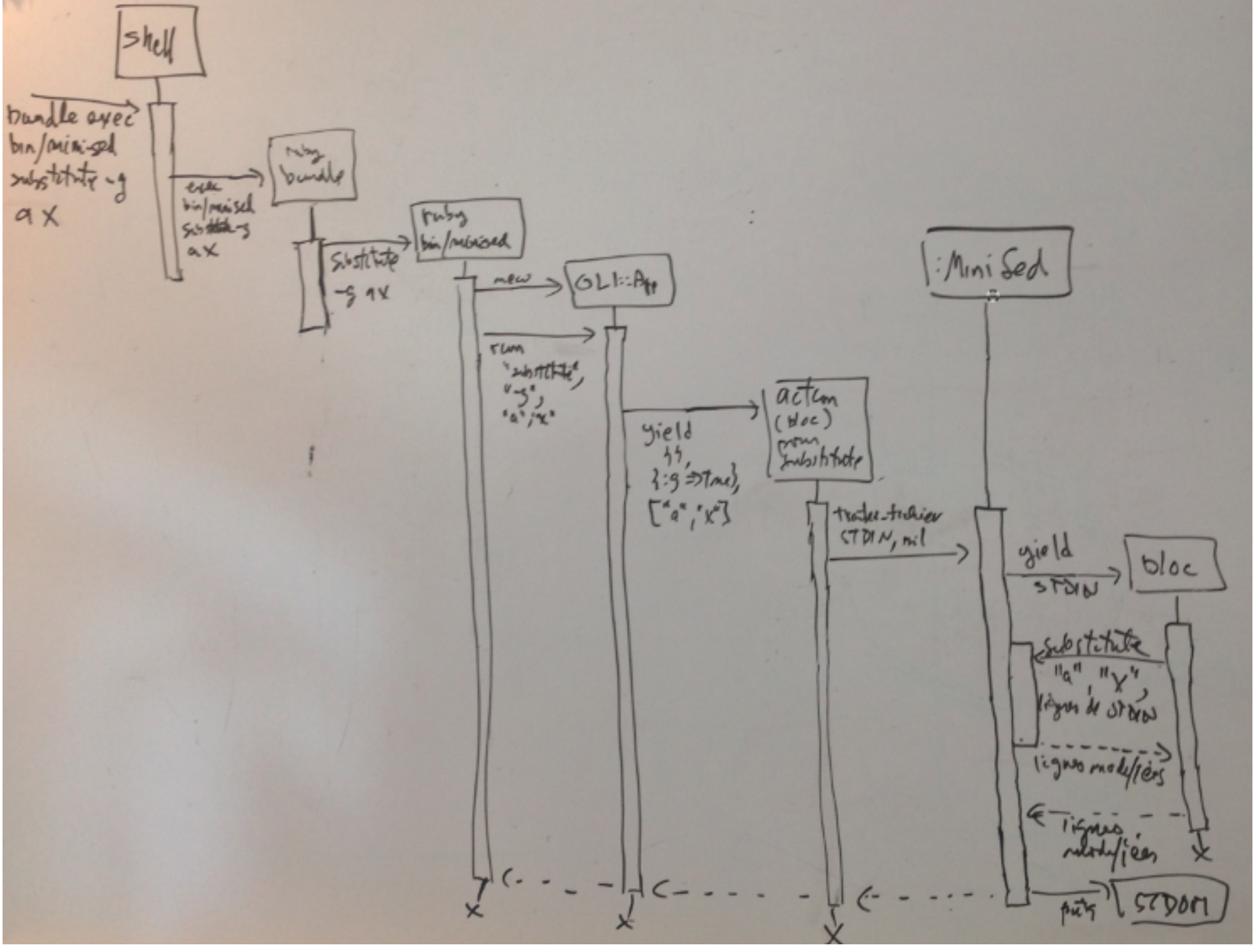
6.9 Comportement dynamique de l’application

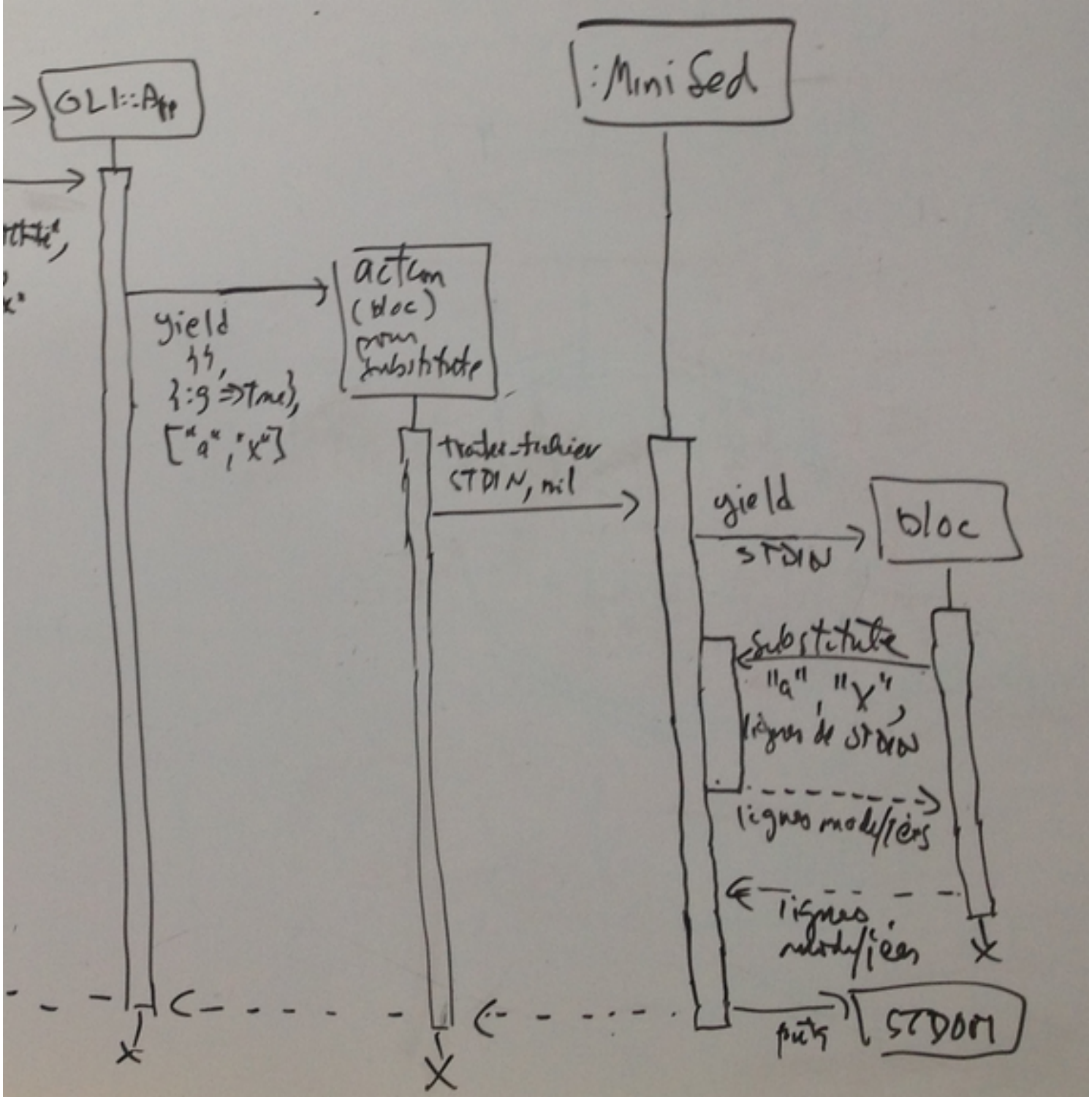
Les différentes étapes de l'exécution d'une commande

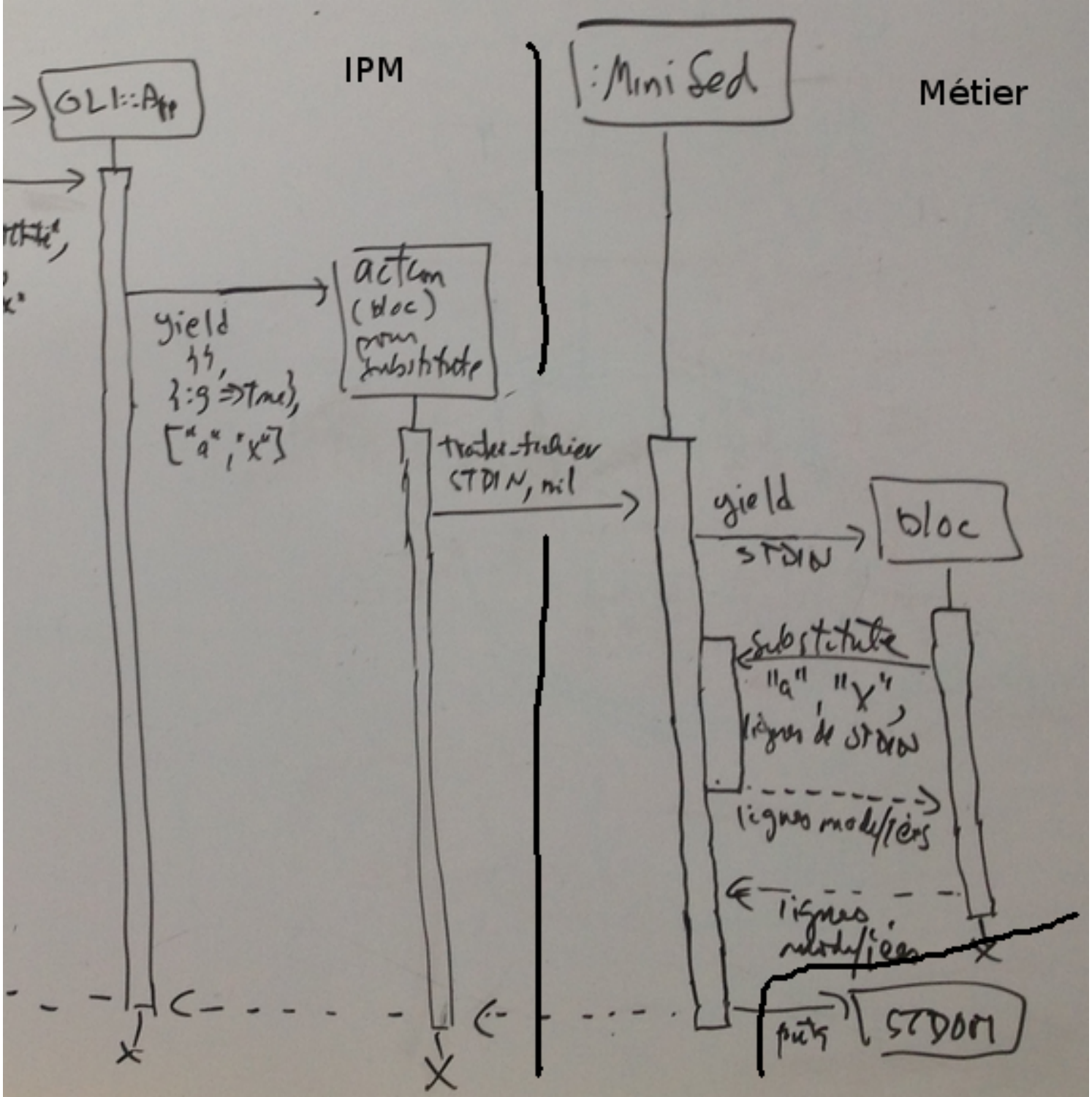
- Les diapositives qui suivent illustrent les différentes étapes de l'exécution d'une commande avec `bundle` pour `mini-sed`

- La commande exécutée dans cet exemple est :

```
$ bundle exec bin/mini-sed substitute -g a X
```


shell
bundle exec bin/mini-sed substitute -g a x
exec bin/mini-sed substitute -g a x
ruby bundle
substitute -g a x
ruby bin/mini-sed
new
GLI::App
run "substitute", "-g", "a", "x"
yield {}, {:g => true}, ["a", "x"]
action (&bloc) pour substitute
:MiniSed
tracer-fichier STDIN, nil
yield STDIN
bloc
substitute "a" "x" lignes de STDIN
lignes modifiées
lignes modifiées
puts
STDOUT


GLI::App
:MiniSed
action
(bloc)
pour
substitute
yield
{:g => true},
["a", "x"]
traiter_fichier
STDIN, nil
yield
STDIN
bloc
substitute
"a" "x"
lignes de STDIN
lignes modifiées
lignes
modifiées
puts
STDOUT


IPM
Métier
:MiniSed
GLI::App
action
(bloc)
pour
substitute
yield
"a",
{:g => true},
["a","x"]
traiter_fichier
STDIN, nil
yield
STDIN
bloc
substitute
"a" "x"
lignes de STDIN
lignes modifiées
lignes
modifiées
puts
STDOUT

Les différentes étapes de l'exécution d'une commande

Au niveau du *shell* :

```
$ bundle exec bin/mini-sed substitute -g a X
```

- Crée un processus (Unix) pour exécuter…

  `ruby bundle exec bin/mini-sed substitute -g a X`

Note : Dans les diapositives qui suivent, l'en-tête du bloc indique les arguments traités par le programme ou la méthode indiqué.

Les différentes étapes de l'exécution d'une commande

Dans `ruby bundle` :

`exec bin/mini-sed substitute -g a X`

- Configure les accès aux *gems* pour `bin/mini-sed` (selon ce qui a été activé préalablement avec `bundle install`).
- Lance l'exécution (commande `exec`) de ...
  `ruby bin/mini-sed substitute -g a X`

Les différentes étapes de l'exécution d'une commande

Dans `ruby bin/mini-sed` :

`substitute -g a X`

- Appelle la méthode . . .

```
GLI::App.run ["substitute", "-g", "a", "X"]
```

Note : Les arguments (tableau de 4 éléments) sont dans `ARGV`.

Les différentes étapes de l'exécution d'une commande

Dans `GLI::App.run` :

```
["substitute", "-g", "a", "X"]
```

- Obtient la description de la commande `substitute`
- Analyse les options et arguments :

```
global_options = {}
options = {:g => true}
args = ["a", "X"]
```

- Exécute l'action (le bloc) associée :

```
c.action do |global_options,options,args|
  motif, chaine, *fichiers = args
  fichiers << STDIN if fichiers.empty?
  fichiers.each do |fichier|
    MiniSed.traiter_fichier( fichier, global_options[:in_p
      MiniSed.substitute( motif, chaine, options[:g], flux
    end
  end
```

Les différentes étapes de l'exécution d'une commande

Dans le bloc pour l'action associée à `substitute` :

```
global_options = {}
options = {:g => true}
args = ["a", "X"]
```

- Décompose les arguments
- Identifie le fichier à traiter comme étant `STDIN`
- Exécute…

```
MiniSed.traiter_fichier( STDIN, nil ) do |flux|
  MiniSed.substitute( motif, chaine, options[:g], flux.rea
end
```

Les différentes étapes de l'exécution d'une commande

Dans `MiniSed.traiter_fichier :`

`fichier = STDIN, ext_in_place = nil`

- Exécute le bloc avec `STDIN`
- Le bloc appelle…

  `MiniSed.substitute("a", "X", true, [`*`lignes`* `STDIN])`

6.10 Encore un peu de *refactoring*

Avec cette solution, il y a moins de code répétitif, mais il y en a encore ☹

```
command :delete do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args
    fichiers << STDIN if fichiers.empty?

    fichiers.each do |fichier|
      MiniSed.traiter_fichier( fichier,
                               global_options[:in_place]
      do |flux|
        MiniSed.delete( motif, flux.readlines )
      end
    end
  end
end
```

Avec cette solution, il y a moins de code répétitif, mais il y en a encore ☹

```
command :print do |c|
  c.action do |global_options,options,args|
    motif, *fichiers = args
    fichiers << STDIN if fichiers.empty?

    fichiers.each do |fichier|
      MiniSed.traiter_fichier( fichier,
                               global_options[:in_place]
      do |flux|
        MiniSed.print( motif, flux.readlines )
      end
    end
  end
end
```

On peut réduire ce code répétitif en introduisant, dans `bin/mini-sed`, une méthode auxiliaire

```
def commande_sans_option commande, nb_arguments: 1
  command commande do |c|
    c.action do |global_options,options,args|
      les_args = args[0...nb_arguments]
      fichiers = args[nb_arguments..-1]
      fichiers << STDIN if fichiers.empty?

      fichiers.each do |fichier|
        in_place = global_options[:in_place]
        MiniSed.traiter_fichier( fichier, in_place ) do |flux|
          MiniSed.send commande, *les_args, flux.readlines
        end
      end
    end
  end
end
```

On utilise ensuite cette méthode auxiliaire pour définir les commandes

```
desc 'Supprime les lignes contenant un motif'
arg_name 'motif [fichier...]'
commande_sans_option :delete, nb_arguments: 1


desc 'Imprime les lignes contenant un motif'
arg_name 'motif [fichier...]'
commande_sans_option :print, nb_arguments: 1
```

Et c'est ensuite (très !) facile de définir de nouvelles commandes

```
desc 'Supprime les lignes contenant un motif'
arg_name 'motif [fichier...]'
commande_sans_option :delete, nb_arguments: 1


desc 'Imprime les lignes contenant un motif'
arg_name 'motif [fichier...]'
commande_sans_option :print, nb_arguments: 1


desc 'Insertion d\'une ligne devant une qui matche'
arg_name 'motif chaine_a_inserer [fichier...]'
commande_sans_option :insert, nb_arguments: 2
```

6.11 Création d'un *gem* pour distribution

Pour créer un *gem* qu'on pourra distribuer, on doit tout d'abord finaliser le fichier `mini-sed.gemspec`

On ajoute les informations sur l'auteur, la description détaillée du *gem*, etc.

```
$ emacs mini-sed.gemspec
```

```
...
s.author = 'Guy Tremblay'
s.email = 'tremblay.guy@uqam.ca'
s.homepage = 'http://www.labunix.uqam.ca'
s.platform = Gem::Platform::RUBY
s.summary = 'Une version simplifiee de sed'
s.description = 'Une version simplifiee de sed,
                 avec des commandes a la git.
                 Utilise comme labo pour illustrer
                 les gems, dont gli'
...
s.add_development_dependency('rake', '~> 11.1')
...
```

Note : On raffine aussi certaines dépendances, pour éviter les messages d'avertissement...

On crée le *gem* avec la commande `gem build`

Il manque la licence d'utilisation... donc à compléter, mais autrement, le *gem* peut être distribué

```
$ ls -o mini-sed-0.0.1.gem
ls: mini-sed-0.0.1.gem: No such file or directory

$ gem build mini-sed.gemspec
WARNING:  licenses is empty, but is recommended. Use a l
WARNING:  See http://guides.rubygems.org/specification-r
  Successfully built RubyGem
  Name: mini-sed
  Version: 0.0.1
  File: mini-sed-0.0.1.gem

$ ls -o mini-sed-0.0.1.gem
-rw-r--r-- 1 tremblay 13824 [...] mini-sed-0.0.1.gem
```

Le *gem* peut alors être distribué, installé, etc.

Pas de documentation `ri` pour ce *gem*, donc on utilise l'option «`--no-ri`»

```
$ gem list mini-sed

*** LOCAL GEMS ***


$ gem install --no-ri mini-sed-0.0.1.gem
Successfully installed mini-sed-0.0.1
1 gem installed


$ gem list mini-sed

*** LOCAL GEMS ***

mini-sed (0.0.1)
```

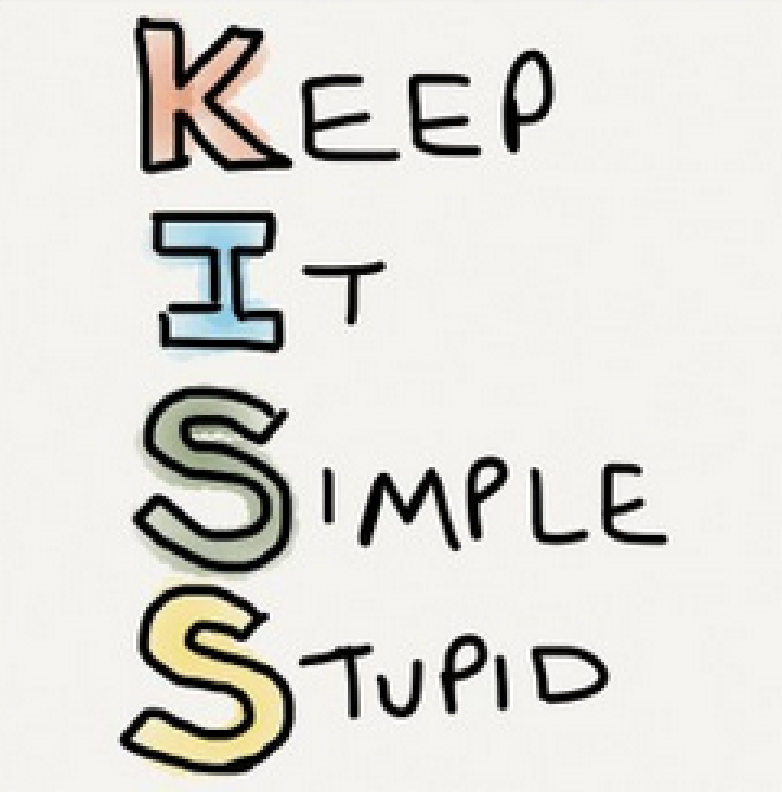
6.12 Conclusion sur l'exemple `mini-sed`

Quelques pratiques à ne pas oublier

- Quand nos tests s'exécutent avec succès, on n'a pas encore terminé : il faut revoir le code et l'améliorer
  $\Rightarrow$ *Refactoring* !

Quelques pratiques à ne pas oublier

- Quand nos tests s'exécutent avec succès, on n'a pas encore terminé : il faut revoir le code et l'améliorer
  $\Rightarrow$ *Refactoring* !



Source: `http://user47329.vs.easily.co.uk/tag/kiss-keep-it-simple-stupid/`

Source: `https://programmingenthusiast.wordpress.com/2015/05/08/`

Quelques pratiques à ne pas oublier

- Quand nos tests s'exécutent avec succès, on n'a pas encore terminé : il faut revoir le code et l'améliorer
  $\Rightarrow$ *Refactoring* !

- Un des effets du *refactoring* est souvent d'introduire des méthodes auxiliaires.

- Si on définit des méthodes auxiliaires, alors on doit définir des tests unitaires pour ces méthodes.

Quelques pratiques à ne pas oublier

- Quand nos tests s'exécutent avec succès, on n'a pas encore terminé : il faut revoir le code et l'améliorer
  $\Rightarrow$ *Refactoring* !

- Un des effets du *refactoring* est souvent d'introduire des méthodes auxiliaires.

- Si on définit des méthodes auxiliaires, alors on doit définir des tests unitaires pour ces méthodes.

- Et quand vous avez progressé, alors **faites un *commit* !**

Références

G.T. Brown.

Ruby Best Practices.

O'Reilly, 2009.

D.B. Copeland.

Build Awesome Command-Line Applications in Ruby : Control Your Computer, Simplify Your Life.

The Pragmatic Bookshelf, 2012.

A. Quelques exemples d'utilisation du «vrai» `sed`

La commande sed

Quelques exemples de substitution

```
$ cat fich.txt
xxx yyy zzz
d/e/f
```

Substitutions simples

```
$ sed 's/x/abc/' fich.txt
abcxx yyy zzz
d/e/f

$ sed 's/x/abc/g' fich.txt
abcabcabc yyy zzz
d/e/f

$ sed 's/\(.\)\1\1/\1/g' fich.txt
??
```

La commande sed

Quelques exemples de substitution

```
$ cat fich.txt
xxx yyy zzz
d/e/f
```

Substitutions simples

```
$ sed 's/x/abc/' fich.txt
abcxx yyy zzz
d/e/f

$ sed 's/x/abc/g' fich.txt
abcabcabc yyy zzz
d/e/f

$ sed 's/\(.\)\1\1/\1/g' fich.txt
x y z
d/e/f
```

La commande sed

Quelques exemples de substitution

Note : Dans plusieurs cas, les guillemets ne sont pas nécessaires, mais je trouve plus simple de toujours les mettre. . .

```
$ cat fich.txt
xxx yyy zzz
d/e/f
```

Autres substitutions

```
$ sed s/x/abc/g fich.txt
abcabcabc yyy zzz
d/e/f

$ sed 's/\(.\)\1\1/\1/g' fich.txt
x y z
d/e/f

$ sed s/\(.\)\1\1/\1/g fich.txt
xxx yyy zzz
d/e/f
```

Sans `'...'`, la dernière commande est interprétée comme suit ☹

```
sed 's/(.)11/1/g' fich.txt
```

La commande sed

Quelques exemples de substitution avec divers délimiteurs

```
$ cat fich.txt
xxx yyy zzz
d/e/f
```

Substitutions avec d'autres délimiteurs

```
$ sed 's/\/e\//XX/' fich.txt
xxx yyy zzz
dXXf

$ sed 's;/e/;XX;' fich.txt
xxx yyy zzz
dXXf

$ sed 's;/e/;;' fich.txt
xxx yyy zzz
df
```

La commande sed

Quelques exemples avec impression explicite


```
$ cat fich.txt
xxx yyy zzz
d/e/f
```

Impression explicite

```
$ sed '/xx/p' fich.txt
xxx yyy zzz
xxx yyy zzz
d/e/f

$ sed -n '/xx/p' fich.txt
xxx yyy zzz
```

La commande `sed`

Un exemple d'édition «en ligne»

```
$ cat fich.txt
xxx yyy zzz
d/e/f
```

Modification directe du fichier et *backup*

```
$ ls fich*
fich.txt

$ sed -i.bak 's/xxx/123/' fich.txt

$ ls fich*
fich.txt  fich.txt.bak

$ cat fich.txt
123 yyy zzz
d/e/f

$ cat fich.txt.bak
xxx yyy zzz
d/e/f
```

La commande sed

Quelques exemples avec plusieurs expressions


```
$ cat fich.txt
xxx yyy zzz
d/e/f
```

Avec plusieurs commandes/expressions

```
$ sed -e 's/x/A/g' -e 's/A/W/g' fich.txt
WWW yyy zzz
d/e/f


$ cat patrons.txt
s/^xxx \(.*\) \(.*\)$/\2:\1/g
s;/\(e\)/;\1;

$ sed -f patrons.txt fich.txt
zzz:yyy
def
```