

Mailing

Permet d'envoyer le mailing d'un mail complexe à partir d'un simple fichier markdown (avec métadonnée).

En ligne de commande

```
send-mails <fichier md>[ <options>]
```

Options

Description	Options	Notes
Faire une simple simulation de l'envoi	<code>-s</code>	« s » pour « simulation »
Ne pas laisser de délai entre les messages	<code>-d</code>	« d » pour « délai ». Sinon, un temps aléatoire sera laissé entre chaque message envoyé.
Pour afficher l'aide	<code>-h</code>	« h » comme « help », aide en anglais
Ouvrir la version éditable du manuel	<code>-dev</code>	S'emploie donc avec <code>send-mail manuel -dev</code>

Fichiers markdown mailing

Contenu des métadonnées

Les *métadonnées* sont consignées dans le fichier markdown du mailing, en haut, entre deux `---` :

```
---
<métadonnée>
---
<message>
```

Elles sont constituées de paires `Clé = Valeur` ou la clé commence toujours par une capitale suivie de minuscules.

Définitions obligatoires

```
Subject = Le sujet que prendront tous les mails
```

Ce sujet peut contenir du code ruby dans `#{code}` . Voir [Sujet dynamique](#).

```
From = <mail> # adresse mail de l'expéditeur
```

Cette donnée peut avoir la forme `Patronyme <mail valide>`.

```
To = <définition des destinataires>
```

On peut définir les destinataires de plusieurs façons. Voir la [Définition des destinataires](#).

Définitions optionnelles

On peut donner au mailing un nom qui sera utile pour d'autres programmes :

```
Name = Le nom du mailing, juste pour information
```

On peut définir des exclusions avec la clé :

```
Excludes = <définition des exclusions>
```

On peut définir ces exclusions de plusieurs manières, [de la même façon que les destinataires](#). Voir aussi [Définition des exclusions](#).

On peut aussi définir **les images** qui seront utilisées dans le texte. Une image utilisée dans le texte possède toujours un identifiant de la forme `IMG<id>`. Par exemple `IMGlogo`. Dans l'entête, on doit absolument définir son chemin d'accès relatif ou absolu.

```
IMG<id image> = <path absolu ou relatif à l'image>
```

Par exemple, pour le logo qui se trouverait dans un dossier `images` au même niveau que le fichier mailing, on pourrait avoir :

```
---
...
IMGlogo = ./images/logo.jpg
---
Bonjour,

Voici mon logo :

IMGlogo
```

Contenu du message

Le contenu du message est la seconde partie du fichier mailing :

```
---
<métadonnées>
---
<définition des styles>

<message>
```

Voir [Définition du message à envoyer](#).

Définitions

DESTINATAIRES

Les destinataires se définissent dans les [métadonnées] à l'aide des propriétés `To` et `Exclude`. La propriété `To` permet de définir la liste de référence et la propriété `Exclude` permet de retirer des adresses.

Manière de définir destinataires

La propriété `To` peut définir (comme la propriété `Excludes` qu'on verra plus tard) :

1. Une simple adresse avec patronyme : `Patronyme <mail valide>` (dans [une liste](#) s'il y en a plusieurs).
2. Une définition minimale d'un destinataire sous forme de rangée CSV : `"<Patronyme>,<sexe>,<mail>"` (l'ordre n'importe pas). Voir la [Définition sous forme rangée CSV](#).
3. Un chemin d'accès relatif ou absolu à [un fichier CSV valide](#) définissant les adresses et les propriétés des destinataires. C'est la forme qui permet le plus simplement de définir de nombreuses propriétés. Voir [Définition par fichier CSV](#).
4. Une table de type ruby contenant toutes les données à commencer par `{ 'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>" }`. Voir [Définition par table ruby](#).
5. Une liste contenant un ou plusieurs éléments précédents. Voir [Définition dans une liste](#).

Définition des destinataires par rangée CSV

La façon la plus simple, qui ne correspond qu'à un seul destinataire est la suivante :

```
To = "Prénom NOM,H,mail@valide.com"
```

(noter que les guillemets sont optionnels)

Définition des destinataires par fichier CSV

Mais l'utilisation d'un mailing requiert plusieurs destinataires. Le plus simple est de les définir dans [un fichier CSV bien formaté](#) :

```
To = destinataires.csv
```

Avec cette formulation, le fichier `destinataires.csv` doit obligatoirement se situer au même niveau que le fichier mailing. Sinon, on peut indiquer le chemin relatif :

```
To = ../utils/destinataires.csv
```

... ou absolu :

```
To = /Users/phil/mails/destinataires.csv
```

On pourra trouver [ci-dessous le formatage de ce fichier CSV](#).

Définition des destinataires par table ruby

On peut définir les destinataires par :

```
To = {'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>", 'Var1' => "...", 'Var2' => "...", etc. }
```

Les clés doivent être obligatoirement des strings qui commencent par des capitales.

Pour avoir plusieurs destinataires, il suffit de mettre ces tables dans une liste :

```
To = [{'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>"}, {'Mail' => "<mail2>", 'Patronyme' => "<patronyme2>", 'Sexe' => "<H/F>"}, etc., ./fichier.csv]
```

Noter, comme cela est précisé plus haut, qu'il est possible de mettre n'importe quelle donnée dans cette liste.

EXCLUSIONS

Les exclusions peuvent être définies de la [même manière que les destinataires](#).

Exclusions par liste de mails

Le fichier possède le même formatage [que le fichier CSV des destinataires](#).

La seule différence, pour les exclusions, est qu'on n'a besoin que de l'adresse mail. La définition des exclusions peut donc ressembler simplement à :

```
Excludes = ["<mail 1>", "<mail 2>", etc.]
```

Exclusions par filtrage

Une manière puissante d'exclure des destinataires (ou de les choisir précisément) consiste à utiliser un filtre en ruby (donc il est impératif de connaître le langage ruby). On définit le fichier filtre dans la donnée

`Excludes`, seul ou dans une liste :

```
---
...
Excludes = /path/to/filtre.rb
---
```

Ou :

```
---
...
Excludes = ["/path/to/exclusions.csv", "/path/to/filtre.rb"]
```

Noter que ci-dessus, on procèdera d'abord à l'exclusions des mails contenus dans `exclusions.csv` et ensuite on enverra la liste restante à `filtre.rb`. L'ordre, dans `Excludes`, est capital.

Grâce aux listes, on peut enchaîner plusieurs filtres [comme dans l'exemple en annexe](#).

Dans le fichier ruby, on définira une **méthode de classe qui portera le même nom que le fichier** et qui recevra en premier argument la liste actuelles des destinataires (déjà filtrés ou non).

Si on définit le mailing par :

```
---
To = desti.csv
Excludes = ./modules/filtre_femmes.rb
```

... alors on aura un fichier `filtre_femmes.rb` dans le dossier `modules` au même niveau que le fichier mailing qui contiendra :

```
# modules/filtre_femmes.rb
class Mailing

  def filtre_femmes(recipients)
    # Ici le traitement des recipients pour produire
    # la liste recipients_filtred qui sera retournée
    #
    return recipients_filtred
  end
end
```

Paramètres dynamiques

On peut transmettre à la méthode de filtrage d'autres arguments. Il suffit de les mettre à la suite du chemin d'accès au fichier, en indiquant leur type par leur forme (un nombre sera tel quel, une chaîne sera mise entre guillemets, une table sera mise telle quelle, etc.).

```
---  
...  
Excludes = /path/to/mon_filtre.rb <arguments>  
---
```

Ou dans une liste :

```
---  
...  
Excludes = ["out.csv", "filtre.rb <arguments>"]  
---
```

On peut définir par exemple d'envoyer un nombre et une chaîne avec :

```
---  
...  
Excludes = /path/to/mon_filtre.rb 25 "jeudi"  
---
```

Ou dans une liste :

```
---  
...  
Excludes = ["out.csv", "/path/mon_filtre.rb 25 'jeudi'"]  
---
```

Noter les guillemets simples utilisés

On aura alors la définition de la méthode de filtre :

```
# mon_filtre.rb  
class Mailing  
  
  def mon_filtre(recipients, age, jour)  
    recipients.reject do |rec|  
      if rec.age > age  
        false  
      else  
        # On ajoute "(du <jour>)" au patronyme (c'est idiot, juste  
        # pour l'exemple)  
        rec.data['Patronyme'] = rec.data['Patronyme'] + " (du #{jour})"  
        true # le prendre  
      end  
    end  
  end  
end
```

```

        end
      end
    end

end

# Ajout d'une méthode au destinataire
class Receiver
  def age
    Time.now.year - data['Naissance'].to_i
  end
end
end

```

Noter comment nous ajoutons une méthode au destinataire (`class Receiver`) en bas de fichier

Définition dans une liste

Que ce soit les destinataires ou les exclusions, on peut toujours les définir dans une liste. Une liste est définie par des crochets qui doivent commencer et fermer la donnée complète :

```

To = [ <définition par liste> ]
Excludes = [ <définition par liste> ]

```

À l'intérieur de ces crochets, on peut trouver toutes les définitions possibles et mélangées, simples mails, table ruby, fichier CSV, rangée CSV, filtre ruby.

Le seul impératif est d'utiliser des guillemets sur toutes les chaînes. Alors qu'il est possible de définir un destinataire unique sans guillemets, de la façon suivante :

```
To = Patro NYME, H, patronyme@chez.lui
```

... dans une liste, il faudra obligatoirement mettre des guillemets :

```
To = ["Patro NYME, H, patronyme@chez.lui", ...]
```

Noter que c'est la donnée complète qui est entre guillemets, pas ses éléments. Cela permet de distinguer les éléments, qui seront ensuite décomposés.

Idem pour les fichiers :

```
To = /path/to/file.csv
```

Mais dans une liste :

```
To = ["/path/to/file.csv", ...]
```

Mais il ne faut pas mettre de guillemets autour de la définition par table ruby :

```
To = {'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>"}
```

Idem dans une liste :

```
To = [{'Mail' => "<mail>", 'Patronyme' => "<patronyme>", 'Sexe' => "<H/F>"}, ...]
```

Il suffit, en fait, de se souvenir qu'une liste sera évaluée en ruby. Donc tout élément doit être « compréhensible » pour ruby.

Noter que `Excludes` peut être défini aussi par une liste, et que **l'ordre est alors capital**.

MESSAGE

Les styles

Le message, deuxième partie du fichier mailing, peut commencer par une définition des styles qui seront utilisés ensuite. Ces styles sont définis en haut du fichier et doivent tous commencer par un point suivi du nom du sélecteur (attention de ne pas confondre avec l'utilisation pure en CSS où les points sont utilisés pour les `class`). Ici, le point définit un sélecteur. Par exemple, si on a cette définition :

```
---  
# Métadonnées  
---  
.mabalise {color: red;}
```

... alors on utilisera dans le message :

```
---  
# Métadonnées  
---  
.mabalise {color: red;}  
  
Bonjour,  
  
Ce texte sera en <mabalise>rouge</mabalise>.
```

... qui produira le message :

«
Bonjour,
Ce texte sera en rouge.
»

Le pseudo-format markdown

Ensuite, le message lui-même peut utiliser un marquage proche de markdown, mais simplifié. Il peut faire usage :

Description	Marquage	Notes
Italiques	<code>*...*</code>	
Gras	<code>**...**</code>	
Souligné	<code>~...~</code>	
Listes	<code>* <item></code> <code>* <item></code> <code>* <item></code>	
Liste numérotée	<code>1. \<item></code> <code>* <item></code> <code>* <item></code>	Contrairement à markdown le premier item doit obligatoirement être <code>1.</code> et les autres items doivent être des astérisques.
Titres	<code># titre</code> , <code>## titre</code> , etc.	Pour définir précisément leur style, définir <code>.h1</code> , <code>.h2</code> , <code>.h3</code> etc. en haut du fichier. Les points, avec les « h », sont obligatoires, contrairement à CSS.

Les images

On peut ensuite utiliser des images, en les définissant dans l'entête.

Noter dès à présent qu'on peut définir très précisément la position et la taille d'une image en utilisant les [styles](#) comme c'est montré [dans l'annexe](#).

Fichier CSV des destinataires

Un fichier destinataires bien formaté doit obligatoirement posséder une entête définissant les champs et des virgules pour séparer les données.

Le nom des champs doit être capitalisé et comprendre au minimum `Patronyme` (avec le nom en capitales), `Mail` et `Sexe` (de valeur `H` ou `F`). **L'ordre des champs n'importe pas.**

Les autres champs seront des données qu'on pourra reprendre à l'aide de variables `%{Variable}` dans le texte du message.

Donc, un fichier minimum ressemblera à :

```
Patronyme,Mail,Sexe
Prénom NOM,sonmail@chez.lui,H
Prénome NOM,autremail@chez.elle,F
```

Ce format vaut aussi bien pour les fichiers de destinataires que les fichiers d'exclusion.

Annexes

Exemples de code

Sujet dynamique

```
Subject = Le sujet du #{Time.now.wday}
```

Images avec styles

Grâce aux [styles](#), on peut définir très précisément la taille et la position des images dans le fichier. Par exemple, ci-dessous, on crée une rangée de vignettes de livres :

```
---  
...  
IMGlivre1 = ./images/livre1.jpg  
IMGlivre2 = ./images/livre2.jpg  
IMGlivre3 = ./images/livre3.jpg  
---  
.vignette {display: inline-block; width:120px;margin:10px;}  
  
Bonjour,  
  
Voici nos derniers livres :  
  
<vignette>IMGlivre1</vignette><vignette>IMGlivre2</vignette>  
<vignette>IMGlivre3</vignette>  
  
Bien à vous.
```

Cela produit le mail suivant, avec les trois images sur la même ligne, de taille 120 pixels en largeur et séparés de 20 pixels.

Noter que les trois vignettes doivent s'inscrire impérativement sur la même ligne (sans retour chariot). Et si possible, sans espace entre les balises.

Chaine de filtres

Grâce aux listes, on peut enchaîner plusieurs listes. Par exemple :

```

---
...
To = destinataires.csv
Excludes = ["filtre_hommes.rb", "filtre_age.rb 25"]
---

Bonjour %{Patronyme},

Vous êtes une femme âgée de plus de 25 ans.

```

Le mailing ci-dessus utilise la liste des destinataires `destinataires.csv` qui définit les propriétés normales ainsi que la propriété `Naissance` de l'année de naissance :

```

Id,Patronyme,Mail,Sexe,Naissance
1,Lui NOM,sonmail@chez.lui,H,2000
2,Elle NOM,sonadresse@chez.elle,F,2003
etc.

```

Le premier filtre va exclure tous les hommes, il est défini ainsi :

```

# filtre_hommes.rb
class Mailing
  def self.filtre_hommes(destinataires)
    destinataires.reject do |r|
      r.sexe == 'H'
    end
  end
end

```

Le second filtre va ne prendre que les destinataires restants de plus de 25 ans. Noter que cette méthode prend en second argument le nombre 25.

```

# filtre_age.rb
class Mailing
  def self.filtre_age(destinataires, age_minimum)
    # On définit l'année courante pour pouvoir calculer
    # l'âge
    current_year = Time.new.year
    # On boucle sur les destinataires pour les filtrer
    destinataires.reject do |r|
      # On calcule l'âge du destinataires
      age = current_year - r.data['Naissance'].to_i
      age < age_minimum
    end
  end
end

```