

Aide-mémoire pour IFT-1902

Gabriel Crépeault-Cauchon

20 décembre 2017

Table des matières

1	Introduction	3
2	Terminal	3
2.1	Commandes de base	3
2.2	Fonctions utiles du terminal et les Regex	3
2.2.1	Expressions régulières (<i>Regex</i>)	3
2.2.2	<code>grep</code>	4
2.2.3	<code>awk</code>	4
2.3	Utiliser Git sur le terminal	4
2.3.1	Configuration	4
2.3.2	Collaborer sur un projet	5
3	Programmation en R	5
3.1	Commandes et expressions de R	5
3.1.1	Commandes de base	5
3.1.2	Opérateurs de R	6
3.2	Objets de R	6
3.2.1	Information sur un objet de R	6
3.2.2	Mode d'un objet	6
3.2.3	NA, NaN, NULL et Inf	8
3.3	Vecteurs	8
3.3.1	Création d'un Vecteur	8
3.3.2	Ajout d'étiquettes au vecteur	8
3.3.3	Indiçage du vecteur	9
3.3.4	Opérations sur les vecteurs	9
3.4	Matrices	9
3.4.1	Création d'une matrice	9
3.4.2	informations supplémentaires sur notre matrice	10
3.4.3	Tableau	10
3.4.4	Indiçage d'une matrice	11
3.4.5	Opérations sur des matrices	11
3.4.5.1	Concaténation de matrices	11
3.4.5.2	Mathématiques / Statistiques	11
3.5	Listes	12
3.5.1	Construction d'une liste	12
3.5.2	Indiçage d'une liste	12
3.5.3	Défaire une liste	13
3.6	Data frame	13
3.6.1	construire un Data frame	13
3.6.2	fonction subset	13
3.7	Importation et exportation de données	14
3.7.1	Exportation	14
3.7.2	Importation	14
3.8	Fonctions internes de R	15
3.8.1	Création de données	15

3.8.1.1	Séquence	15
3.8.1.2	Répétition	15
3.8.1.3	Simulation d'un échantillon	16
3.8.1.4	Séquence de lettres	16
3.8.2	Triage de données	16
3.8.3	Recherche et position	16
3.8.4	Tests logiques	17
3.8.5	Arrondir une valeur	17
3.8.6	Statistiques	17
3.8.7	Extraction d'information	19
3.8.8	les fonctions "is.blablabla"	19
3.8.9	Fonctions d'applications	20
3.8.10	Algorithmes	20
3.8.10.1	Algorithme de tri	20
3.8.10.2	Algorithme de recherche	22

1 Introduction

Section plus qualitative à compléter plus tard

2 Terminal

2.1 Commandes de base

Voici quelques commandes essentielles à connaître pour naviguer terminal Bash :

Important : certaines fonctions du *Shell* demandent un espace, d'autre des tirets.

- `cd` pour *change directory* (nous permet de changer de dossier dans le terminal) ;
- `pwd` pour savoir le chemin d'accès dans lequel on se trouve en ce moment ;
- `ls` pour faire apparaître la liste de tous les fichiers dans le dossier actuel ;
- `ls -a` fait apparaître **tous les dossiers**, même ceux qui commence par un point (exemple, *.Renviron*) ;
- `touch` Créer un fichier de texte brut ;
- `rm` permet
- `mkdir` Créer un dossier dans le répertoire où l'on se trouve ;
- `rmdir` Supprimer un dossier dans le répertoire où l'on se trouve ;
- `mv <nomfichier> <destination>` : dépalacer un fichier
- `nano <nomfichier>` : éditeur de texte intégré à Bash pour modifier un fichier de texte brut (pas très convial, mais permet de faire rapidement des petites modifications).
- Pour sortir de l'éditeur *nano*, on fait **Ctrl+X**

2.2 Fonctions utiles du terminal et les Regex

2.2.1 Expressions régulières (*Regex*)

Voici un aide-mémoire sur les expressions régulières

Expression	définition	Exemples
<code>*</code>	Cherche 1 ou plusieurs occurrences du caractère précédent	<code>ga*</code> va trouver <code>ga</code> , <code>g</code> et <code>gaaaa</code>
<code>?</code>	Cherche 1 ou 0 occurrences du caractère précédent	<code>ga</code> va trouver <code>ga</code> et <code>g</code> , mais pas <code>gaaaa</code>
<code>+</code>	Cherche 1 ou plusieurs du caractère précédent	<code>ga+</code> va trouver
<code>strut</code>	Quand on cherche un caractère spécial (utilisé dans les <i>Regex</i>)	<code>Hungry\?</code> va trouver <code>Hungry?</code>
<code>.</code>	Cherche n'importe quel caractères	<code>ga.</code> va trouver <code>gab</code> , <code>garage</code> , <code>gabon</code> , etc.
<code>()</code>	cherche une chaîne de caractères	
<code>[]</code>	cherche parmi une liste de caractères	<code>[gb]ateaux</code> va trouver <code>gateaux</code> et <code>bateaux</code>
<code> </code>	Va chercher la chaîne de caractères avant ou après le symbole	<code>(lun) (mar)di</code> va trouver <code>lundi</code> et <code>mardi</code>

Expression	définition	Exemples
{ }	Spécifie le nombre (consécutif) d'occurrence	
^	le caractère doit se trouver au début de la ligne	<code>^http</code> nous permet de trouver des URL
\$	le caractère doit terminer la ligne	<code>(.com)\$</code> nous permet de trouver les adresses internet se terminant par .com

2.2.2 grep

2.2.3 awk

```
awk -F "," '{print $1 " " $5}' IAG.TO.csv
```

```
## Date Close
## 2017-11-17 59.990002
## 2017-11-20 59.900002
## 2017-11-21 60.330002
## 2017-11-22 60.189999
## 2017-11-23 59.700001
## 2017-11-24 59.669998
## 2017-11-27 59.380001
## 2017-11-28 59.049999
## 2017-11-29 59.139999
## 2017-11-30 60.169998
## 2017-12-01 59.889999
## 2017-12-04 60.270000
## 2017-12-05 60.279999
## 2017-12-06 59.860001
## 2017-12-07 59.459999
## 2017-12-08 60.000000
## 2017-12-11 59.570000
## 2017-12-12 59.650002
## 2017-12-13 59.779999
## 2017-12-14 58.820000
## 2017-12-15 58.980000
```

2.3 Utiliser Git sur le terminal

2.3.1 Configuration

À la première utilisation de git via le terminal *Bash* il faut configurer quelques informations :

- `git config --global user.name "<Nom>"` Configurer son nom tel qu'on désire qu'il apparaisse sur Git.
- `git config --global user.email "<courriel>"` Configurer l'adresse courriel associée.
- `git config --global core.editor open` Si vous oubliez de préciser une description lors d'un *commit* (sera vu à la prochaine section), il va simplement ouvrir un fichier texte brut.

- `git config --list` Juste pour valider que les informations entrées ci-dessus sont enregistrées adéquatement.

2.3.2 Collaborer sur un projet

Pour faire le suivi des versions d'un projet informatique, il est utile d'utiliser Git. Voici un résumé des fonctions (du terminal) à savoir utiliser :

- `git init` : Créer un répertoire (*repository*) *Git* Dans le dossier actuel. *Astuce* : il est plus simple de créer son *repository* directement sur Github puis le cloner dans le dossier désiré;
- `git clone <https://...>` *Cloner* (ou si on préfère, télécharger) un répertoire Git dans le dossier actuel. **Attention**, on va cloner une seule fois un répertoire, car par la suite on va *pull* les modifications du dépôt;
- `git pull` commandes qu'on utilise seulement si on a déjà cloné le répertoire. Nous permet d'avoir les mises à jour;
- `git status` permet de voir si il y a des fichiers dans notre dossier qui n'apparaissent pas (ou que les modifications n'apparaissent pas) sur le répertoire Git. Si c'est le cas, elles seront affichés en **rouge**.
- `git add` pour ajouter les modifications dans le dépôt. Après cette étape, on doit confirmer nos modifications par la commande *commit*
 - Si on utilise `git add -A`, tous les fichiers seront ajoutés au prochain *commit*
- `git commit -m "<description de la modif.>"` On confirme notre modification au travail et on décrit *très brièvement* ce qu'on a fait comme modification
- `git push pousser` au serveur les modifications qu'on a fait. Après cette étape, si on va sur Github, nos modifications apparaîtront.
- `git log --oneline --decorate` : permet de voir les dernières modifications qu'il y a eu sur le projet **et Git nous indique dans quelle branche du dépôt on se trouve**
- `git branch` : identifie toutes les branches existantes pour le projet en cours
 - `git branch <nom_nouvelle_branche>` : créer une nouvelle branche indépendante
 - `git branch -d {nom_branche_à_supprimer}` : supprimer une branche qu'on n'a plus de besoin
 - `'git branch --no-merged` : visualiser les branches dans lesquelles il y a des modifications qui n'ont pas encore été *merged*.
- `git checkout <nom_de_la_branche>` : pour se déplacer d'une branche à l'autre.
- `git merge <nom_branche>` : faire un *merge* entre la branche master et la branche indépendante pour fusionner les modifications.

3 Programmation en R

3.1 Commandes et expressions de R

3.1.1 Commandes de base

- `save image()` Si on veut sauvegarder l'espace de travail et son environnement. **Rarement utilisée**, sauf si on veut sauvegarder la valeur d'une variable (qui est longue à obtenir)
- `getwd()` obtenir le répertoire de travail dans lequel on se trouve actuellement
- `setwd(<chemin d'accès>)` Changer le répertoire de travail actuel
- `help()` obtenir de l'aide sur une fonction ou une commande en particulier. On peut aussi accéder au manuel d'instruction de R avec `help.start`
- `ls()` : voir tous les objets de l'environnement global
- `rm()` : pour supprimer un objets
- `rm(list = ls())` : supprimer tous les objets dans l'environnement global

3.1.2 Opérateurs de R

Opérateur	Fonction
\$	extraction d'une liste
[]	indiciage
^	puissance
-	changement de signe
:	génération d'une suite
%% %/%	produit matriciel, modulo, division entière
* /	multiplication, division
+ -	addition, soustraction
< <= == >= > !=	plus petit, plus petit ou égal, égal, plus grand ou égal, plus grand, différent
!	de négation logique
&	ET logique
	OU logique
<-	affectation (méthode la plus utilisée)

3.2 Objets de R

3.2.1 Information sur un objet de R

- `mode()` : Mode d'un objet
- `length()` : Longueur d'un objet
- `nchar()` : nombre de caractères
- `class()` : classe d'un objet
- `summary()` : beaucoup d'information sur l'objet

3.2.2 Mode d'un objet

Mode	Contenu de l'objet
numeric	nombres réels
complex	nombres complexes
logical	valeurs booléennes
character	chaînes de caractères
function	fonction
list	liste
expression	expressions non évaluées

```
char <- c("a", "b", "c")
mode(char)
```

```
## [1] "character"
```

```
num <- c(1:5)
mode(num)
```

```
## [1] "numeric"
```

Si on crée un vecteur qui contient des données de plus d'un mode, il va convertir les autres données dans le mode le plus «puissant», en respectant l'ordre suivant :

1. list
2. character
3. numeric
4. logical

```
a <- c(TRUE, "test", 1:2, list(1)) ; a
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] "test"
##
## [[3]]
## [1] 1
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 1
```

```
mode(a)
```

```
## [1] "list"
```

```
b <- c(FALSE, 0:2, "test") ; b
```

```
## [1] "FALSE" "0"      "1"      "2"      "test"
```

```
mode(b)
```

```
## [1] "character"
```

```
c <- c(FALSE, 1:2) ; c
```

```
## [1] 0 1 2
```

```
mode(c)
```

```
## [1] "numeric"
```

```
x <- c(TRUE, 1, 4, "GAB")
class(x)
```

```
## [1] "character"
```

```
mode(x)
```

```
## [1] "character"
```

3.2.3 NA, NaN, NULL et Inf

```
0/0
```

```
## [1] NaN
```

```
Inf/Inf
```

```
## [1] NaN
```

```
Inf-Inf
```

```
## [1] NaN
```

```
1/0
```

```
## [1] Inf
```

```
Inf
```

```
## [1] Inf
```

```
Inf^Inf
```

```
## [1] Inf
```

```
5 + NULL
```

```
## numeric(0)
```

3.3 Vecteurs

3.3.1 Création d'un Vecteur

```
(x <- c(1,2,3))
```

```
## [1] 1 2 3
```

```
(y <- vector(mode = "numeric", length = 5))
```

```
## [1] 0 0 0 0 0
```

3.3.2 Ajout d'étiquettes au vecteur

il existe 2 façon :

```
x <- c(1,2)
```

```
names(x) <- c("a","b") ; x
```

```
## a b
```

```
## 1 2
```

ou bien

```
x <- c(a=1, b=2) ; x
```

```
## a b
```

```
## 1 2
```


3.3.3 Indigage du vecteur

```
x <- c(A = 1, B = 2, C = 3, D = 4, E = TRUE, G = NA) ; x
```

```
## A B C D E G
## 1 2 3 4 1 NA
```

```
x[3] # 3e élément
```

```
## C
## 3
```

```
x[c(1,5)] # 1er et 5e élément
```

```
## A E
## 1 1
```

```
x[-3] # tout sauf le 3e élément
```

```
## A B D E G
## 1 2 4 1 NA
```

```
x[-c(3,4)] # tous sauf le 3e et 4e élément
```

```
## A B E G
## 1 2 1 NA
```

```
x[!is.na(x)] # tout ce qui n'est pas NA
```

```
## A B C D E
## 1 2 3 4 1
```

```
x[x<4] # tous les éléments qui sont <4
```

```
## A B C E <NA>
## 1 2 3 1 NA
```

```
x[c("G","B")] # seulement les éléments taggés "G" et "B"
```

```
## G B
## NA 2
```

```
x[] # tous les éléments
```

```
## A B C D E G
## 1 2 3 4 1 NA
```

3.3.4 Opérations sur les vecteurs

3.4 Matrices

3.4.1 Création d'une matrice

On peut créer une matrice avec la fonction `matrix` :

```
x <- (matrix(5:8, nrow = 2, ncol = 2, byrow = T))
```

L'option `byrow` est pour que la matrice se remplisse par ligne.

```
diag(2)           # Créé une matrice identité p*p

##      [,1] [,2]
## [1,]    1    0
## [2,]    0    1

diag(c(5,4))      # matrice carrée avec le vecteur en argument

##      [,1] [,2]
## [1,]    5    0
## [2,]    0    4
```

3.4.2 informations supplémentaires sur notre matrice

```
attributes(x)

## $dim
## [1] 2 2

nrow(x)           # nombre de lignes

## [1] 2

ncol(x)           # nombre de colonnes

## [1] 2
```

3.4.3 Tableau

C'est comme une matrice, mais à plus de 2 dimensions. Le deuxième argument devient alors un vecteur d'au moins 3 dimensions.

```
(tableau <- array(1:30, dim = c(2,5,3)))

## , , 1
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
##
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   11   13   15   17   19
## [2,]   12   14   16   18   20
##
## , , 3
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   21   23   25   27   29
## [2,]   22   24   26   28   30

?array
```

3.4.4 Indigage d'une matrice

Comme un vecteur, on peut indiquer une matrice.

```
x[1]           # 1er élément

## [1] 5

x[1,2]         # 1ère ligne, 2e colonne

## [1] 6

x[2,]         # toute la 2e ligne

## [1] 7 8

x[,1]         # toute la 1ère colonne

## [1] 5 7

x[c(1,2),2]   # les 2 premiers éléments de la première colonne

## [1] 6 8
```

3.4.5 Opérations sur des matrices

3.4.5.1 Concaténation de matrices

```
rbind(x,x[1,]*x[2,])  # ajout de ligne à la matrice

##      [,1] [,2]
## [1,]    5    6
## [2,]    7    8
## [3,]   35   48

cbind(x,5)           # ajout d'une colonne à la matrice

##      [,1] [,2] [,3]
## [1,]    5    6    5
## [2,]    7    8    5
```

3.4.5.2 Mathématiques / Statistiques

Lorsqu'on a un vecteur ou une matrice, on peut utiliser des fonctions qui font certains calculs par ligne ou par colonne :

```
(tableau <- matrix(1:9, nrow = 3, ncol = 3, byrow = T))

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9

rowSums(tableau)     # somme de chaque ligne

## [1]  6 15 24

colSums(tableau)     # somme de chaque colonne

## [1] 12 15 18
```

```

rowMeans(tableau)  # moyenne de chaque ligne

## [1] 2 5 8
colMeans(tableau)  # moyenne de chaque colonne

## [1] 4 5 6
det(tableau)       # calcul le déterminant

## [1] 6.661338e-16
t(x)              # pour avoir la matrice transposée

##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
solve(x)          # nous donne la matrice inverse de x

##      [,1] [,2]
## [1,] -4.0  3.0
## [2,]  3.5 -2.5
solve(x, c(1,2))  # pour des équations linéaires ...

## [1]  2.0 -1.5

```

3.5 Listes

3.5.1 Construction d'une liste

```

(sondage <- list(nom = c("Justin","William","Charlie"),
  age = c(35,23,14),
  job = c("Politicien","Menuisier","Etudiant"),
  citoyen = c(T,T,F)))

## $nom
## [1] "Justin" "William" "Charlie"
##
## $age
## [1] 35 23 14
##
## $job
## [1] "Politicien" "Menuisier" "Etudiant"
##
## $citoyen
## [1] TRUE TRUE FALSE

```

3.5.2 Indigage d'une liste

```

sondage[2]        # 2e élément de la liste

## $age
## [1] 35 23 14

```

```

sondage$age           # idem

## [1] 35 23 14
# pour pouvoir travailler avec les données, il faut faire un double indigage :
mean(sondage[2])

## Warning in mean.default(sondage[2]): argument is not numeric or logical:
## returning NA

## [1] NA
mean(sondage[[2]])

## [1] 24
sondage[[c(1,2)]]     # 2e élément du 1er élément de la liste

## [1] "William"

```

3.5.3 Défaire une liste

On peut *défaire* une liste avec `unlist`. La conversion se fait vers le mode le plus puissant.

3.6 Data frame

3.6.1 construire un Data frame

Avec la fonction `dataframe`, on peut faire quelque

```

(form <- data.frame(prenom = c("Suzie", "Mario", "Jean"),
  nom = c("Tremblay", "Gagnon", "Cote"),
  age = c(12, 13, 14),
  fumeur = c(T, T, F),
  salaire = c(40000, 45000, 135000)))

##   prenom      nom age fumeur  salaire
## 1 Suzie Tremblay 12  TRUE   40000
## 2 Mario  Gagnon 13  TRUE   45000
## 3  Jean   Cote 14 FALSE  135000

```

3.6.2 fonction subset

La fonction `subset` permet d'extraire de l'information dans le data frame en appliquant un filtre.

```

subset(form, salaire > 42000, select = age)

##   age
## 2  13
## 3  14

```

3.7 Importation et exportation de données

3.7.1 Exportation

```
# Pour la fonction cat, on peut ajouter plusieurs objets de R dans la concaténation
cat(num,"Ceci est un commentaire", file = "sondage.data")
# La fonction write n'accepte qu'un seul objet (ou matrice)
write(tableau, file = "sondage.data", ncolumns = 5)
# spécifiquement pour exporter en .csv (virgule)
write.csv(tableau,file = "sondage.csv")
# encore en .csv (mais séparé par des points-virgules)
write.csv2(char, file = "caracteres.csv2")
```

3.7.2 Importation

```
scan("sondage.data")    # pour importer des données
```

```
## [1] 1 4 7 2 5 8 3 6 9
```

```
read.table(file = "IAG.TO.csv",header = T, sep = ",")
```

```
##           Date  Open  High   Low Close Adj.Close Volume
## 1  2017-11-17 60.46 60.56 59.86 59.99  59.61127 154000
## 2  2017-11-20 59.88 60.17 59.82 59.90  59.52183 168400
## 3  2017-11-21 60.00 60.50 59.85 60.33  59.94912 153700
## 4  2017-11-22 60.39 60.64 59.85 60.19  59.81000 152200
## 5  2017-11-23 60.01 60.01 59.47 59.70  59.70000  57800
## 6  2017-11-24 59.87 60.12 59.61 59.67  59.67000  70100
## 7  2017-11-27 59.70 59.97 59.36 59.38  59.38000 103800
## 8  2017-11-28 59.37 59.62 58.67 59.05  59.05000 192000
## 9  2017-11-29 59.20 59.76 59.05 59.14  59.14000 159900
## 10 2017-11-30 59.38 60.69 58.90 60.17  60.17000 367900
## 11 2017-12-01 60.19 60.31 59.24 59.89  59.89000 156000
## 12 2017-12-04 60.00 60.52 59.83 60.27  60.27000 189200
## 13 2017-12-05 60.35 61.02 60.08 60.28  60.28000 170400
## 14 2017-12-06 60.14 60.19 59.44 59.86  59.86000 148100
## 15 2017-12-07 59.85 59.91 59.31 59.46  59.46000 101000
## 16 2017-12-08 59.58 60.18 59.45 60.00  60.00000  90200
## 17 2017-12-11 59.99 60.07 59.42 59.57  59.57000  87000
## 18 2017-12-12 59.52 59.94 59.31 59.65  59.65000  91900
## 19 2017-12-13 59.61 60.16 59.60 59.78  59.78000 113500
## 20 2017-12-14 59.78 59.85 58.78 58.82  58.82000  90900
## 21 2017-12-15 59.07 59.53 58.92 58.98  58.98000 236645
```

```
read.csv(file = "IAG.TO.csv")    # Comprends par défaut qu'il y a un titre
```

```
##           Date  Open  High   Low Close Adj.Close Volume
## 1  2017-11-17 60.46 60.56 59.86 59.99  59.61127 154000
## 2  2017-11-20 59.88 60.17 59.82 59.90  59.52183 168400
## 3  2017-11-21 60.00 60.50 59.85 60.33  59.94912 153700
## 4  2017-11-22 60.39 60.64 59.85 60.19  59.81000 152200
## 5  2017-11-23 60.01 60.01 59.47 59.70  59.70000  57800
## 6  2017-11-24 59.87 60.12 59.61 59.67  59.67000  70100
## 7  2017-11-27 59.70 59.97 59.36 59.38  59.38000 103800
```

```
## 8  2017-11-28 59.37 59.62 58.67 59.05  59.05000 192000
## 9  2017-11-29 59.20 59.76 59.05 59.14  59.14000 159900
## 10 2017-11-30 59.38 60.69 58.90 60.17  60.17000 367900
## 11 2017-12-01 60.19 60.31 59.24 59.89  59.89000 156000
## 12 2017-12-04 60.00 60.52 59.83 60.27  60.27000 189200
## 13 2017-12-05 60.35 61.02 60.08 60.28  60.28000 170400
## 14 2017-12-06 60.14 60.19 59.44 59.86  59.86000 148100
## 15 2017-12-07 59.85 59.91 59.31 59.46  59.46000 101000
## 16 2017-12-08 59.58 60.18 59.45 60.00  60.00000  90200
## 17 2017-12-11 59.99 60.07 59.42 59.57  59.57000  87000
## 18 2017-12-12 59.52 59.94 59.31 59.65  59.65000  91900
## 19 2017-12-13 59.61 60.16 59.60 59.78  59.78000 113500
## 20 2017-12-14 59.78 59.85 58.78 58.82  58.82000  90900
## 21 2017-12-15 59.07 59.53 58.92 58.98  58.98000 236645
```

On aurait pu aussi utiliser `read.csv2()` si on a un jeu de données où les champs sont séparés par des points virgules.

3.8 Fonctions internes de R

Les fonctions présentées ci-dessous sont livrées de base dans R.

3.8.1 Création de données

3.8.1.1 Séquence

```
seq(from = 10, to = 20, by = 2)
```

```
## [1] 10 12 14 16 18 20
```

```
x <- c(1,2,3,8)
seq(x)
```

```
## [1] 1 2 3 4
```

```
seq_len(5)
```

```
## [1] 1 2 3 4 5
```

```
y <- c(10,14,3,2)
seq_along(y)
```

```
## [1] 1 2 3 4
```

3.8.1.2 Répétition

```
rep(1:2, each = 5)
```

```
## [1] 1 1 1 1 1 2 2 2 2 2
```

```
rep(1:2, times = 5) # remarquez la différence avec 'each'
```

```
## [1] 1 2 1 2 1 2 1 2 1 2
```

```
rep.int(1:3,7) # Respecte le nombre de répétitions désirées
```

```
## [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
```

```
rep_len(1:3,10)      # Arrête le vecteur à une certaine longueur
```

```
## [1] 1 2 3 1 2 3 1 2 3 1
```

```
# (même si ça ne respecte pas la longueur du vecteur)
```

3.8.1.3 Simulation d'un échantillon

Note : si le nombre de résultats possibles est différent de la taille de l'échantillon, l'option `replace=TRUE` doit être activée (tirage avec remplacement)

```
x <- sample(1:5, size = 8, replace = TRUE, prob = c(0.1,0.2,0.2,0.25,0.25)) ; x
```

```
## [1] 2 5 5 3 5 2 3 3
```

3.8.1.4 Séquence de lettres

```
x <- c(1,2,3)
```

```
letters[x]          # en minuscule
```

```
## [1] "a" "b" "c"
```

```
x <- c(24,25,26)
```

```
LETTERS[x]          # EN MAJUSCULE
```

```
## [1] "X" "Y" "Z"
```

3.8.2 Triage de données

```
sort(c(-4,0,5,2,60,1), decreasing = F)      # trie les données
```

```
## [1] -4  0  1  2  5 60
```

```
rank(c(-4,0,5,2,60,1))      # donne le rang croissant/décroissant
```

```
## [1] 1 2 5 4 6 3
```

```
rev(1:5)                   # créé un vecteur renversé
```

```
## [1] 5 4 3 2 1
```

```
unique(c(1,1,1,2,4,5,6,6,6))      # renvoie une seule fois chaque éléments du vecteur
```

```
## [1] 1 2 4 5 6
```

3.8.3 Recherche et position

Lorsqu'on recherche un élément en particulier dans un vecteur ou qu'on veut filtrer l'information :

```
which(x<8)                # position des éléments dans le vecteur qui respecte la condition
```

```
## integer(0)
```

```
which.min(x)              # position du minimum
```

```
## [1] 1
```



```
which.max(x)      # position du maximum

## [1] 3
match(19,x)       # cherche le premier argument dans x

## [1] NA
-1:3 %in% x       # Valide si -1,0,1,2 ou 3 est présent dans le vecteur x (booléen)

## [1] FALSE FALSE FALSE FALSE FALSE
```

3.8.4 Tests logiques

```
any(x > 5)        # est-ce qu'il y a un élément de x > 5 ?

## [1] TRUE
all(x > 7)        # est-ce que TOUS les éléments sont plus grands que 7 ?

## [1] TRUE
```

3.8.5 Arrondir une valeur

```
(precise <- c(1.445346, 3.345345435, 4.4346345, 78.345363, -6.780597345))

## [1] 1.445346 3.345345 4.434634 78.345363 -6.780597
round(precise)     # arrondi à l'entier près

## [1] 1 3 4 78 -7
round(precise, 3)  # on précise qu'on veut 3 décimales

## [1] 1.445 3.345 4.435 78.345 -6.781
floor(precise)     # arrondi à l'entier inférieur

## [1] 1 3 4 78 -7
ceiling(precise)   # arrondi à l'entier supérieur

## [1] 2 4 5 79 -6
trunc(precise)     # on enlève les décimales (sans arrondir, différent de floor pour les négatifs)

## [1] 1 3 4 78 -6
```

3.8.6 Statistiques

Il existe plusieurs fonctions déjà installées pour des calculs statistiques simples :

```
(echant_x <- sample(1:20, 10, replace=T))

## [1] 14 3 6 7 17 12 13 11 6 2
(echant_y <- sample(1:30, 10, replace = T))
```

```
## [1] 6 11 21 30 3 7 2 25 10 8
sum(echant_x)      # Somme d'un vecteur

## [1] 91
prod(echant_x)     # produit de tous les éléments d'un vecteur

## [1] 617512896
mean(echant_x)     # Moyenne

## [1] 9.1
var(echant_x)      # Variance

## [1] 24.98889
cov(echant_x,echant_y) # Covariance (nécessite de mettre 2 vecteurs en argument)

## [1] -17.14444
min(echant_x)      # minimum

## [1] 2
max(echant_x)      # maximum

## [1] 17
median(echant_x)   # médiane

## [1] 9
quantile(echant_x) # quantile

##      0%    25%    50%    75%   100%
##    2.00    6.00    9.00   12.75   17.00
diff(echant_x)     # delta (variation) entre chaque éléments

## [1] -11    3    1   10   -5    1   -2   -5   -4
range(echant_x)    # étendue

## [1] 2 17
summary(echant_x)  # résumé des principales mesures statistiques

##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      2.00    6.00    9.00    9.10   12.75   17.00
```

Il est parfois pratique de connaître les résultats cummulatifs :

```
cumsum(echant_y)   # somme cumulative

## [1] 6 17 38 68 71 78 80 105 115 123
cumprod(echant_y)  # produit cumulatif

## [1] 6 66 1386 41580 124740 873180
## [7] 1746360 43659000 436590000 3492720000
cummin(echant_y)   # minimum cumulatif

## [1] 6 6 6 6 3 3 2 2 2 2
```

```

cummax(echant_y)      # maximum cumulatif

## [1]  6 11 21 30 30 30 30 30 30 30

pmin(echant_y, 12)     # minimum entre : valeur en argument et l'élément du vecteur

## [1]  6 11 12 12  3  7  2 12 10  8

pmax(echant_y, 9)      # maximum entre : valeur en argument et l'élément du vecteur

## [1]  9 11 21 30  9  9  9 25 10  9

```

3.8.7 Extraction d'information

```

head(echant_x, n = 4)  # on extrait les n premiers éléments de l'objet

## [1] 14  3  6  7

head(echant_x, n = -4) # l'objet R mais sans les 4 derniers éléments

## [1] 14  3  6  7 17 12

tail(echant_x, n = 3)  # on extrait les n derniers éléments de l'objet

## [1] 11  6  2

tail(echant_x, n = -2) # l'objet $ mais sans les 2 premiers éléments

## [1]  6  7 17 12 13 11  6  2

```

3.8.8 les fonctions "is.blablabla"

Il y a plusieurs fonctions dans R qui permettent d'obtenir une réponse booléenne **vrai** ou **faux** en validant une information.

```

is.matrix(y)          # une liste n'est pas une matrice

## [1] FALSE

is.array(y)           # comme on voit, une matrice est aussi un tableau

## [1] FALSE

is.function(y)        # Est-ce que l'argument est une fonction?

## [1] FALSE

is.character(char)    # de mode "character"?

## [1] TRUE

is.numeric(num)       # de mode "numeric"?

## [1] TRUE

is.vector(y)          # est-ce que c'est un vecteur?

## [1] TRUE

```

3.8.9 Fonctions d'applications

Lorsqu'on travaille avec des *data frame* ou bien des tableaux dans R, on est mieux d'utiliser des fonctions d'applications (plutôt que des boucles qui ralentissent le temps d'exécution d'une fonction).

- le premier argument est le vecteur ou le tableau sur lequel on veut *appliquer* la fonction
- le 2e argument est la dimension (1 = ligne, 2= colonne, etc.) sur laquelle on veut appliquer itérativement la fonction
- La fonction désirée sur chaque ligne ou colonne est donnée en 3e argument
- si la fonction a besoin de plusieurs arguments de spécifiés, on peut les spécifier par la suite en 4e, 5e argument etc...

Exemples

```
apply(tableau, 1, sum)    # fait la somme des lignes
```

```
## [1]  6 15 24
```

```
apply(tableau, 2, mean)   # idem pour colonne
```

```
## [1]  4 5 6
```

3.8.10 Algorithmes

3.8.10.1 Algorithme de tri

3.8.10.1.1 Insertionsort

$$runtime = O(N^2)$$

```
insertionsort <- function(x)
{
  xlen <- length(x)
  for(i in seq_len(xlen))
  {
    for (j in seq_len(i-1))
    {
      if (x[j] > x[i])
        x <- c(x[seq_len(j-1)],
               x[i],x[j],
               x[j + seq_len(i-j-1)],
               x[i + seq_len(xlen-i)])
    }
  }
  x
}
```

3.8.10.1.2 Selectionsort

$$runtime = O(N^2)$$

```

selectionsort <- function(x)
{
  xlen <- length(x)      # sert souvent
  for (i in seq_len(xlen))
  {
    ## recherche de la position de la plus petite valeur
    ## parmi x[i], ..., x[xlen]
    i.min <- i
    for (j in i:xlen)
    {
      if (x[j] < x[i.min])
        i.min <- j
      ## à mesure que i augmente, j ne considère pas
      ## les première position, puisqu'elles sont déjà triées
    }
    ## échange de x[i] et x[i.min]
    x[c(i, i.min)] <- x[c(i.min, i)]
  }
  x
}

```

3.8.10.1.3 Bubblesort

$$runtime = O(N^2)$$

```

bubblesort <- function(x)
{
  ind <- 2:length(x)     # suite sert souvent

  not_sorted <- TRUE     # entrer dans la boucle
  while (not_sorted)
  {
    not_sorted <- FALSE
    for (i in ind)
    {
      j <- i - 1
      if (x[i] < x[j])
      {
        x[c(i, j)] <- x[c(j, i)]
        not_sorted <- TRUE
        next
      }
    }
  }
  x
}

```

3.8.10.1.4 Countingsort

$$runtime = O(N + M)$$

```

countingsort <- function(x, min, max)
{
  min1m <- min - 1          # sert souvent
  counts <- numeric(max - min1m) # initialisation

  for (i in seq_along(x))
  {
    j <- x[i] - min1m
    counts[j] <- counts[j] + 1
  }

  ## suite des nombres de 'min' à 'max' répétés chacun le
  ## bon nombre de fois
  rep(min:max, counts)
}

```

3.8.10.1.5 Bucketsort

On place nos données dans des paniers (*buckets*), puis on les place en ordre. Ensuite on remets nos *buckets* dans le bon ordre.

$$runtime = O(N + M)$$

```

bucketsort <- function(data, nbuckets)
{
  # Création des buckets
  max = max(data)
  bucket <- vector(mode = "list", length = nbuckets)
  etendue <- max/nbuckets

  # distribution des données dans les buckets
  for (i in seq_along(data)) # on veut la longueur de data
  {
    j <- ceiling(data[i]/etendue) # pour sélectionner le bon
    bucket[[j]] <- c(bucket[[j]], data[i])
  }

  # tri des différents buckets
  for (i in seq_len(nbuckets)) # on veut le chiffre de la variable
  {
    if (!is.null(bucket[[i]])) # is pas NULL, on le trie!
      bucket[[i]] <- sort(bucket[[i]])
  }

  # On défait la liste pour avoir nos données triées
  unlist(bucket)
}

```

3.8.10.2 Algorithme de recherche

3.8.10.2.1 Linear Search

Cet algorithme va seulement fonctionner si les données sont triées.

$$runtime = O(N)$$

```
linearsearch <- function(x,target)
{
  for (i in seq_along(x))
  {
    if (x[i] == target)
      return(i)      # retourne la position de l'élément recherché
    if (x[i] > target)
      return("la valeur n'est pas dans le vecteur")
  }
  NA
}
```

3.8.10.2.2 Binary Search

$$runtime = O(\log(N))$$

Principe de l'algorithme : on fixe une valeur au milieu. si la valeur recherchée est plus grande, on vient d'éliminer la moitié des données. La recherche va se faire dans la partie supérieure.

```
binary_search <- function(x, target)
{
  min = 1
  max = length(x)

  while (min <= max)
  {
    mid = floor((max + min)/2)    # choisir floor/ceiling arbitraire
    if (target < x[mid])
      max <- mid - 1
    else if (target > x[mid])
      min <- mid + 1
    else return(mid)
  }
  NA      # la valeur qu'on cherche n'est pas dans le vecteur!
}
```

3.8.10.2.3 Interpolation search

va faire un *guess* sur l'endroit approximatif de la valeur, considérant que les données sont triées. Ensuite, il va interpoler dans le bon sens pour se rapprocher (de façon itérative) de la bonne valeur.

$$runtime = O(\log(\log(N)))$$

```
interpolation_search <- function(x, target)
{
  min <- 1
  max <- length(x)
  while (min <= max)
  {
```

```
mid <- min + floor((max - min)*
  (target - x[min]) / (x[max] - x[min]))
if (target < x[mid])
  max <- mid - 1
else if (target > x[mid])
  min <- mid + 1
else
  return(mid)
}
"La valeur cherchée n'est pas dans le vecteur"
}
```