Aide-mémoire pour IFT-1902

Gabriel Crépeault-Cauchon

17 décembre 2017

Table des matières

1 Introduction

Section plus qualitative à compléter plus tard

2 Terminal

2.0.1 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

2.0.2 Utiliser Git sur le terminal

2.0.2.1 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.

2.0.2.2 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*

— `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.

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
`[]`	indiçage
`^`	puissance
`-`	changement de signe
`:`	génération d'une suite
`%*%`	produit ma-
`%%`	tri-
`%/%`	ciel, modulo, division entière
`* /`	multiplication, division
`+ -`	addition, soustraction

Opérateur	Fonction
< <=	plus petit,
==	petit ou égal,
>= >	égal, plus grand ou égal,
!=	plus grand, différent de
!	négation logique
&	ET logique
|	OU logique
<-	affectation (méthode la plus utilisée)

3.1.3 Création d'une suite ou séquence

Séquence de chiffres

```
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
```

Échantillon de données aléatoire

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

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

Séquence de lettres

```
x <- c(1,2,3)
letters[x]
```

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

```
x <- c(24,25,26)
LETTERS[x]
```

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

3.1.4 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 Objets de R

3.2.1 Mode d'un objet

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

Si on créé 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"
```

3.2.2 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)
```