

Chapitre 3 – Les commandes de base

- **A- Commandes de la console**

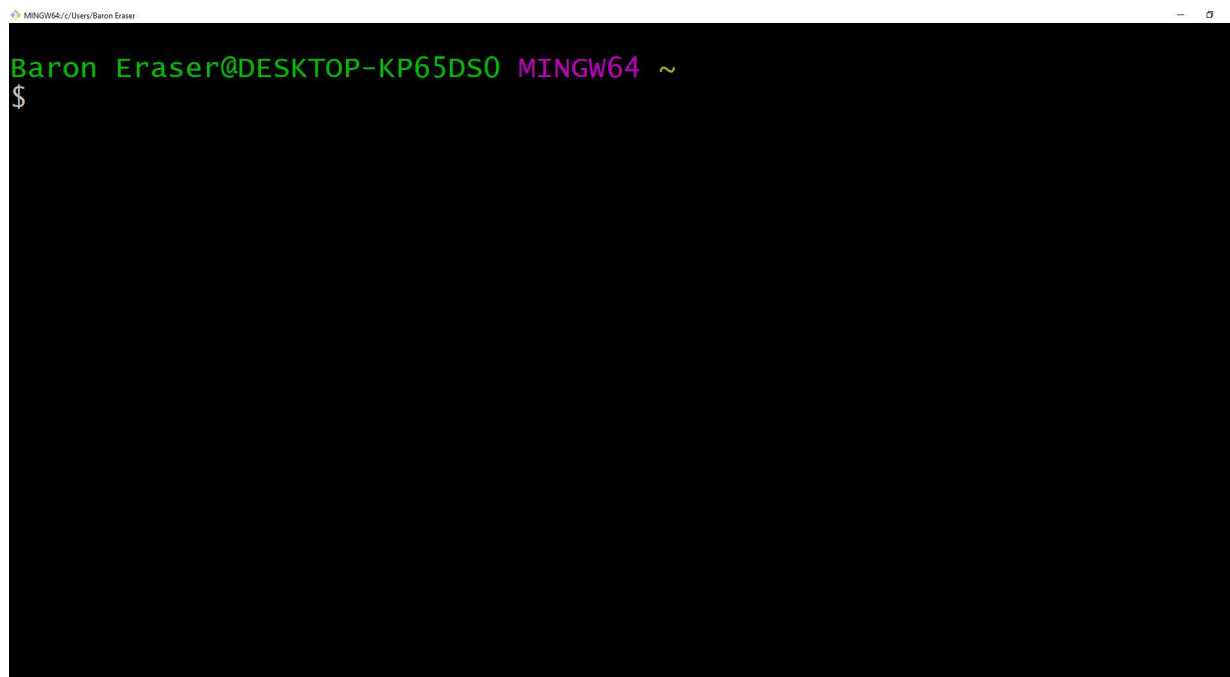
- **La console ?**

La console est un terme désignant l'outil qui permet l'accès au shell du système.

On dit aussi « en ligne de commande ».

Ouvrez GitBash.

Une fois ouvert, vous devriez avoir ceci :

A screenshot of a Git Bash terminal window. The title bar at the top reads "MINGW64/c/Users/Baron Eraser". The terminal has a black background with green text. The prompt is "Baron Eraser@DESKTOP-KP65DS0 MINGW64 ~" followed by a green dollar sign "\$" on the next line, indicating the shell is ready for input.

```
MINGW64/c/Users/Baron Eraser
Baron Eraser@DESKTOP-KP65DS0 MINGW64 ~
$
```

Les commandes de base à savoir pour utiliser la console :

- Lister là où l'on se trouve => ll

```

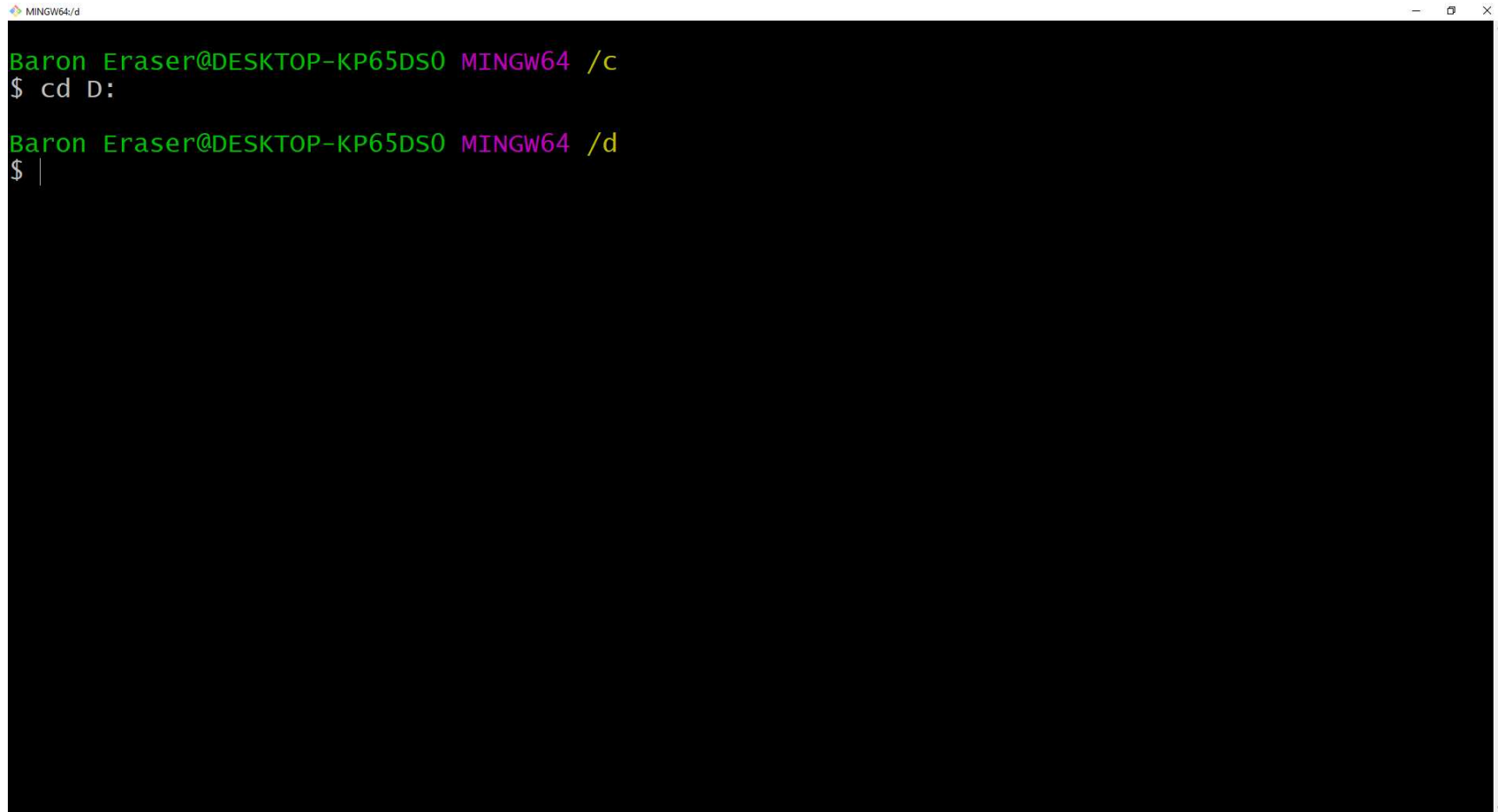
MINGW64/c/Users/Baron Eraser
Baron Eraser@DESKTOP-KP65DS0 MINGW64 ~
$ ll
total 11041
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 '3D Objects'/
drwxr-xr-x 1 Baron Eraser 197121 0 mai 8 08:05 anse/
drwxr-xr-x 1 Baron Eraser 197121 0 août 20 13:03 AppData/
lrwxrwxrwx 1 Baron Eraser 197121 37 août 20 13:03 'Application Data' -> '/c/Users/Baron Eraser/AppData/Roaming/'
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 Contacts/
lrwxrwxrwx 1 Baron Eraser 197121 65 août 20 13:03 Cookies -> '/c/Users/Baron Eraser/AppData/Local/Microsoft/Windows/InetCookies/'
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 23 18:09 Desktop/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 Documents/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 23 21:12 Downloads/
drwxr-xr-x 1 Baron Eraser 197121 0 août 31 17:01 dwhelper/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 Favorites/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 Links/
lrwxrwxrwx 1 Baron Eraser 197121 35 août 20 13:03 'Local Settings' -> '/c/Users/Baron Eraser/AppData/Local/'
lrwxrwxrwx 1 Baron Eraser 197121 66 août 20 13:03 'Menu Démarrer' -> '/c/Users/Baron Eraser/AppData/Roaming/Microsoft/windows/Start Menu/'
lrwxrwxrwx 1 Baron Eraser 197121 31 août 20 13:03 'Mes documents' -> '/c/Users/Baron Eraser/Documents/'
drwxr-xr-x 1 Baron Eraser 197121 0 févr. 28 2019 MicrosoftEdgeBackups/
lrwxrwxrwx 1 Baron Eraser 197121 65 août 20 13:03 Modèles -> '/c/Users/Baron Eraser/AppData/Roaming/Microsoft/windows/Templates/'
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 Music/
-rw-r--r-- 1 Baron Eraser 197121 7864320 sept. 23 21:22 NTUSER.DAT
-rw-r--r-- 1 Baron Eraser 197121 2023424 août 20 13:03 ntuser.dat.LOG1
-rw-r--r-- 1 Baron Eraser 197121 180224 août 20 13:03 ntuser.dat.LOG2
-rw-r--r-- 1 Baron Eraser 197121 65536 août 20 13:03 NTUSER.DAT{0089b0f2-c342-11e9-a7e3-0c9d925dac58}.TM.blf
-rw-r--r-- 1 Baron Eraser 197121 524288 août 20 13:03 NTUSER.DAT{0089b0f2-c342-11e9-a7e3-0c9d925dac58}.TMContainer000000000000000001.regtrans-ms
-rw-r--r-- 1 Baron Eraser 197121 524288 août 20 13:03 NTUSER.DAT{0089b0f2-c342-11e9-a7e3-0c9d925dac58}.TMContainer000000000000000002.regtrans-ms
-rw-r--r-- 1 Baron Eraser 197121 20 août 20 13:08 ntuser.ini
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 16 13:45 OneDrive/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 Pictures/
lrwxrwxrwx 1 Baron Eraser 197121 62 août 20 13:03 Recent -> '/c/Users/Baron Eraser/AppData/Roaming/Microsoft/windows/Recent/'
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 'Saved Games' /
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 13 00:53 Searches/
lrwxrwxrwx 1 Baron Eraser 197121 62 août 20 13:03 SendTo -> '/c/Users/Baron Eraser/AppData/Roaming/Microsoft/windows/SendTo/'
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 07:40 Videos/
drwxr-xr-x 1 Baron Eraser 197121 0 juil. 4 19:25 'VirtualBox VMs'/
lrwxrwxrwx 1 Baron Eraser 197121 73 août 20 13:03 'Voisinage d'impression' -> '/c/Users/Baron Eraser/AppData/Roaming/Microsoft/windows/Printer Shortcuts/'
lrwxrwxrwx 1 Baron Eraser 197121 73 août 20 13:03 'Voisinage réseau' -> '/c/Users/Baron Eraser/AppData/Roaming/Microsoft/windows/Network Shortcuts/'
Baron Eraser@DESKTOP-KP65DS0 MINGW64 ~
$

```

Affichage après avoir tapé la commande ll

- Se déplacer => cd [là ou l'on veut aller]

Exemple : je veux changer de disque => cd D:

A screenshot of a Windows command prompt window titled "MINGW64/d". The prompt shows the user "Baron Eraser@DESKTOP-KP65DS0" in a "MINGW64" environment. The first command entered is "cd D:", and the second command is "cd /d", with a cursor visible after the second command.

```
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /c
$ cd D:

Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d
$ |
```

Affichage après avoir tapé la commande cd D:

- Je retape ll pour lister

```

MINGW64/d
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /c
$ cd D:

Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d
$ ll
total 72
drwxr-xr-x 1 Baron Eraser 197121 0 févr. 28 2019 '$RECYCLE.BIN'/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 23 18:01 '00- Personnel'/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 11 19:29 '01- Musiques Baron Eraser'/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 23 18:04 '02- HashtagFormation'/
drwxr-xr-x 1 Baron Eraser 197121 0 août 1 11:02 '03- Safed'/
drwxr-xr-x 1 Baron Eraser 197121 0 juil. 16 16:00 '04- Videos perso'/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 20 10:26 '07- Boite mail'/
drwxr-xr-x 1 Baron Eraser 197121 0 mars 2 2019 '11- Projets videos'/
drwxr-xr-x 1 Baron Eraser 197121 0 juil. 3 12:48 '14- Programmes'/
drwxr-xr-x 1 Baron Eraser 197121 0 juin 11 11:08 '15- Films et Programmes ftp'/
drwxr-xr-x 1 Baron Eraser 197121 0 mars 2 2019 '16- FirefoxProfile'/
drwxr-xr-x 1 Baron Eraser 197121 0 mars 2 2019 '17- ThuderbirdProfile'/
drwxr-xr-x 1 Baron Eraser 197121 0 mars 7 2019 FondDecran/
drwxr-xr-x 1 Baron Eraser 197121 0 mai 19 07:32 Jeux/
drwxr-xr-x 1 Baron Eraser 197121 0 août 20 13:59 Recovery/
drwxr-xr-x 1 Baron Eraser 197121 0 mars 2 2019 Steam/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 18 09:35 'System volume Information'/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 5 15:05 WorkspaceYaum/
drwxr-xr-x 1 Baron Eraser 197121 0 juil. 16 15:47 'zzz- archives'/

Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d
$ |

```

Affichage après avoir tapé la commande ll sur le disque D:

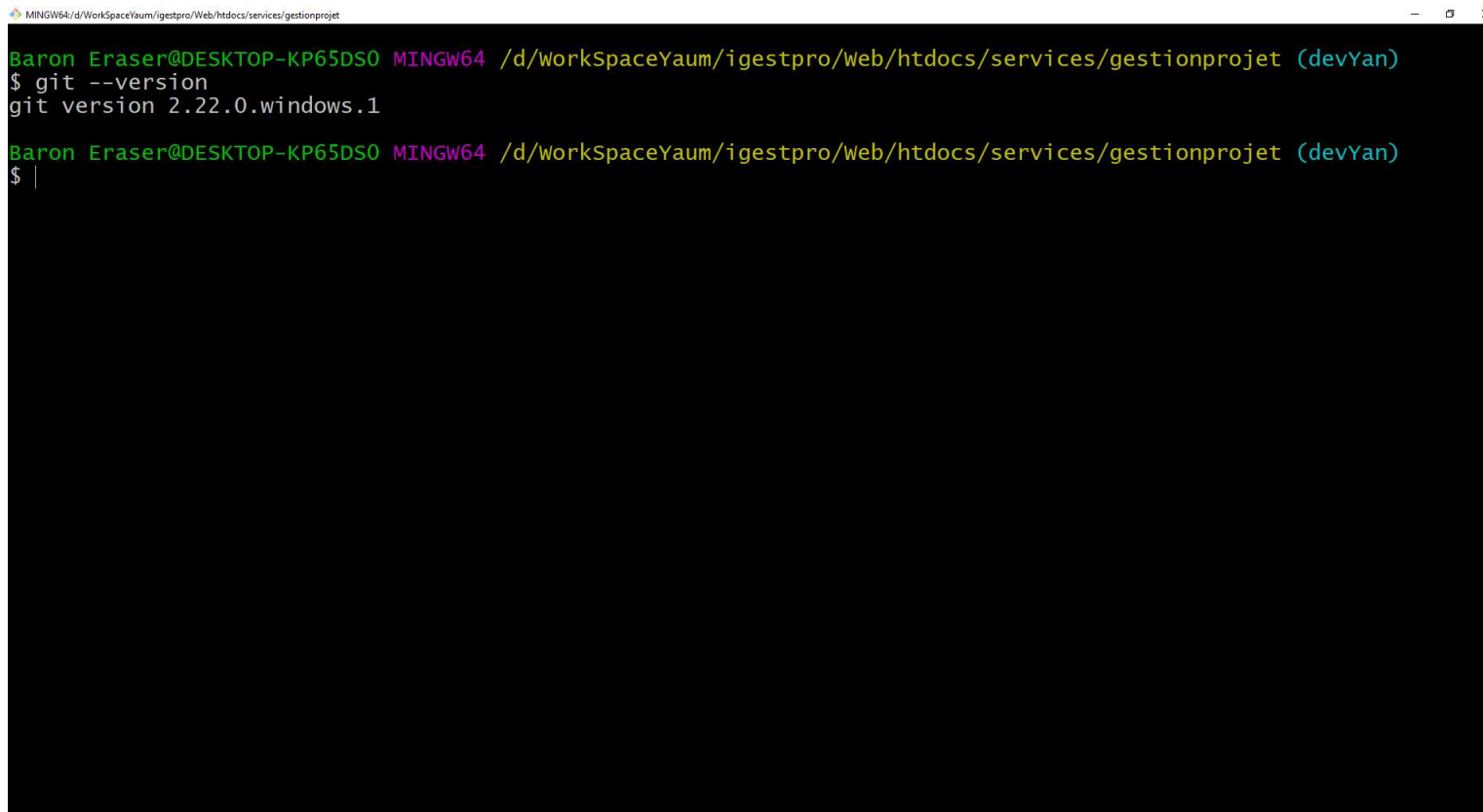
On ne va lister toutes les commandes, cela ne serait pas pertinent. En voici les principales dont nous aurons besoin :

- Lister => ll ou ls
- Se déplacer => cd nomdisque ou nomdossier
- Se déplacer => sortir du dossier =>: cd ..
- Lister en triant par ordre des dernières modifications +ancien vers + récent=> ls -ltr
- Lister en triant par ordre des dernières modifications +récent vers + ancien=> ls -lt
- Copier un fichier => cp nomDuFichier nomDuNouveauFichier
- Supprimer un fichier => rm nomDuFichier
- Créer un dossier => mkdir nomDuDossier
- Supprimer un dossier => rm -rf nomDuDossier
- Afficher ou vous vous trouver => pwd
- Nettoyer votre affichage de toutes vos commandes => clear
- Voir l'historique de toutes vos commandes => history

- Appeler GIT ?

Pour ce faire c'est très simple : on appelle toujours le programme en premier :

■ git --version

A screenshot of a Windows command prompt window with a black background and green text. The window title is "MINGW64/d:/WorkSpaceYaum/igestpro/Web/htdocs/services/gestionprojet". The prompt shows the user "Baron Eraser@DESKTOP-KP65DS0" in a "MINGW64" environment at the path "/d:/workSpaceYaum/igestpro/Web/htdocs/services/gestionprojet". The command "git --version" has been entered and executed, resulting in the output "git version 2.22.0.windows.1". The prompt "\$" is visible on the next line.

```
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/workSpaceYaum/igestpro/Web/htdocs/services/gestionprojet (devYan)
$ git --version
git version 2.22.0.windows.1
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/workSpaceYaum/igestpro/Web/htdocs/services/gestionprojet (devYan)
$ |
```

Affichage après avoir tapé la commande git --version

Maintenant passons une commande erronée :

- `git --café`

A screenshot of a Windows terminal window with a black background and green text. The window title is 'MINGW64: /d/WorkSpaceYaum/igestpro/Web/htdocs/services/gestionprojet'. The prompt is 'Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/WorkSpaceYaum/igestpro/Web/htdocs/services/gestionprojet (devYan)'. The first command is '\$ git --version', which outputs 'git version 2.22.0.windows.1'. The second command is '\$ git --café', which outputs 'unknown option: --café' followed by the usage information for the git command.

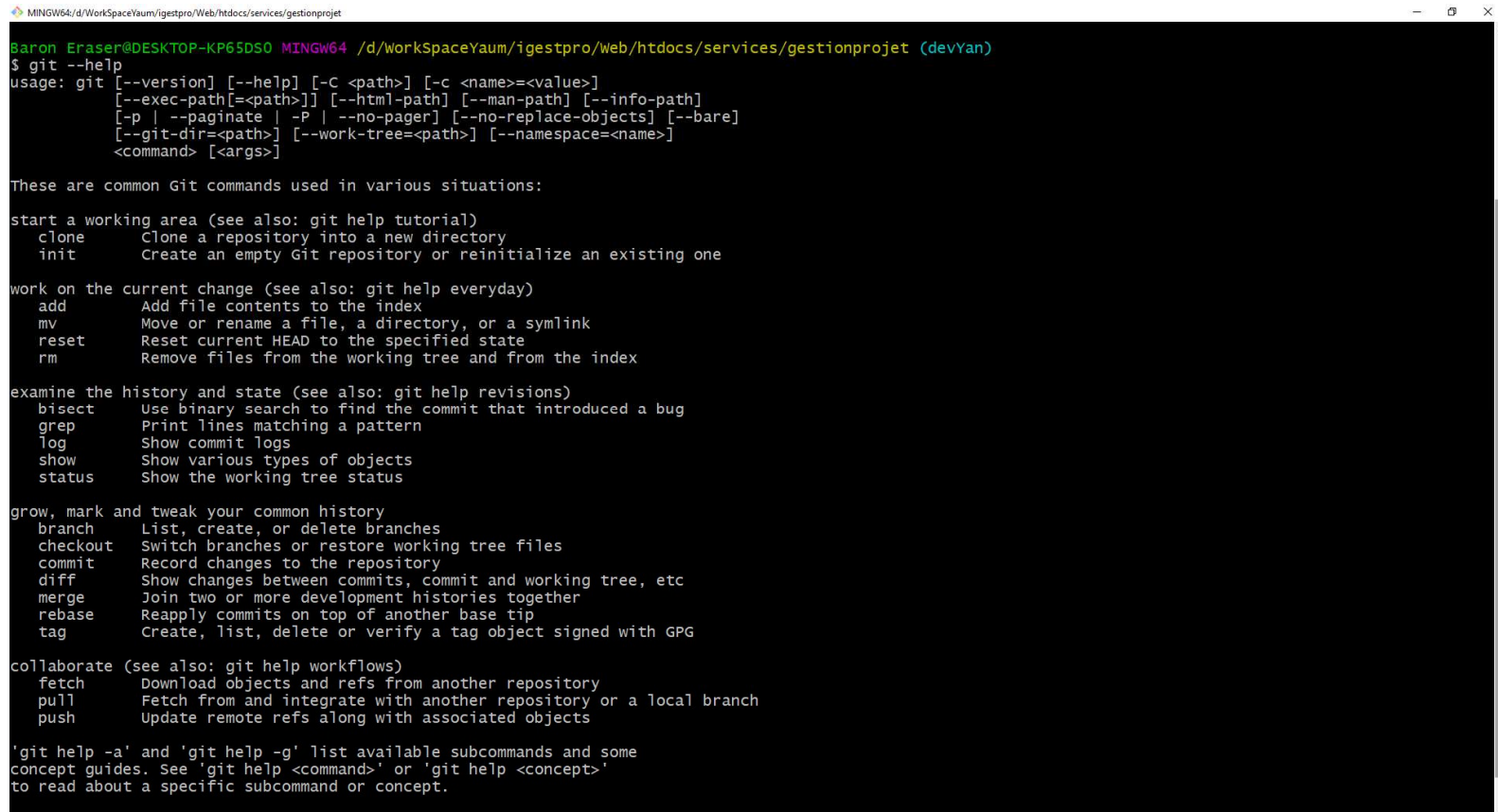
```
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/WorkSpaceYaum/igestpro/Web/htdocs/services/gestionprojet (devYan)
$ git --version
git version 2.22.0.windows.1

Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/WorkSpaceYaum/igestpro/Web/htdocs/services/gestionprojet (devYan)
$ git --café
unknown option: --café
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <command> [<args>]
```

Affichage après avoir tapé la commande `git --café`

Tentons la commande :

- `git --help`



```
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/WorkSpaceYaum/igestpro/Web/htdocs/services/gestionprojet (devYan)
$ git --help
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
        [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
        [-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]
        [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
        <command> [<args>]

These are common Git commands used in various situations:


start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv         Move or rename a file, a directory, or a symlink
  reset      Reset current HEAD to the specified state
  rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect     Use binary search to find the commit that introduced a bug
  grep       Print lines matching a pattern
  log        Show commit logs
  show       Show various types of objects
  status     Show the working tree status

grow, mark and tweak your common history
  branch     List, create, or delete branches
  checkout   Switch branches or restore working tree files
  commit     Record changes to the repository
  diff       Show changes between commits, commit and working tree, etc
  merge      Join two or more development histories together
  rebase     Reapply commits on top of another base tip
  tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch      Download objects and refs from another repository
  pull       Fetch from and integrate with another repository or a local branch
  push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

Affichage après avoir tapé la commande `git --help`

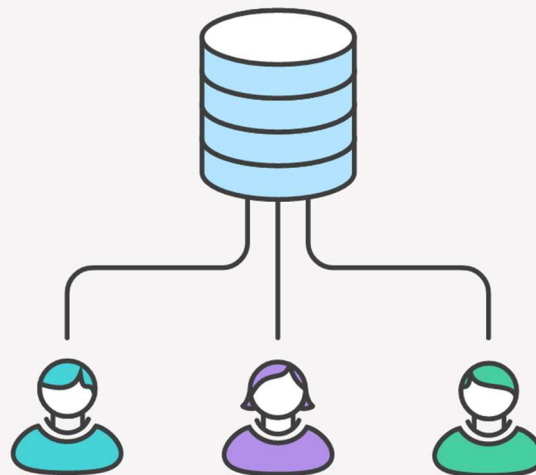
On peut voir que git a reconnu la commande et nous à retourner les informations concernant l'aide pour l'utiliser.

- **B- Cloner**

- **git clone**

La commande git clone sert en fait à englober plusieurs autres commandes. Elle crée un nouveau dossier, va à l'intérieur de celui-ci et lance git init pour en faire un dépôt Git vide, ajoute un serveur distant (git remote add) à l'URL que vous lui avez passée (appelé par défaut origin), lance git fetch à partir de ce dépôt distant et ensuite extrait le dernier commit dans votre répertoire de travail avec git checkout.

Everybody clones the central repository



Vous clonez un dépôt avec git clone [url].

Commençons par nous positionner à l'endroit où vous souhaitez cloner notre projet de travail collaboratif : coursGit sur GitLab.

Notes : qu'est-ce que GitLab ?

Gitlab, c'est une plateforme permettant d'héberger et de gérer des projets web de A à Z. Présentée comme la plateforme des développeurs modernes, elle offre la possibilité de gérer ses dépôts Git et ainsi de mieux appréhender la gestion des versions de vos codes sources.



- **TD - git clone**

Donc, comme je disais, commençons par nous positionner à l'endroit où vous souhaitez cloner notre projet de travail collaboratif : coursGit sur GitLab.

- Ouvrez la console, créez un dossier que vous appelez workSpaceCoursGit (pour espace de travail).
 - `cd D :`
 - `mkdir workSpaceCoursGit`
 - `cd workSpaceCoursGit`

- Donc, une fois que nous avons créé notre dossier et que nous nous y sommes placé à l'intérieur :
 - `git clone https://gitlab.com/micheligestpro/coursgit.git`

Vous devriez voir ceci :

```
Cloning into 'coursgit'...
remote: Enumerating objects: 26, done.
remote: Counting objects: 100% (26/26), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 26 (delta 14), reused 0 (delta 0)
Unpacking objects: 100% (26/26), done.
```

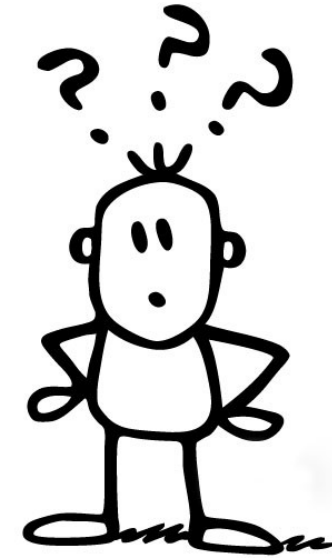
Si vous faites ll, vous devriez voir ceci :

```
$ ll
total 4
drwxr-xr-x 1 votre_nom 197121 0 sept. 24 15:41 coursgit/
```

Si vous faites cd coursgit/ et ensuite ll vous devriez voir ceci :

```
MINGW64:/d/WorkSpaceYaum/workSpaceCoursGit/coursgit
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/WorkSpaceYaum/workSpaceCoursGit/coursgit (master)
$ ll
total 2
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 archives/
-rw-r--r-- 1 Baron Eraser 197121 21 sept. 24 15:41 fichiersInutiles.txt
-rw-r--r-- 1 Baron Eraser 197121 48 sept. 24 15:41 lienOriginal.txt
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie1/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie2/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie3/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie4/
```

Que s'est-il passé ?



On peut voir déjà dans la barre d'adresse qu'il est apparu entre parenthèse le terme (master) :

MINGW64:/d/WorkSpaceYaum/workSpaceCoursGit/coursgit

```
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/workSpaceYaum/workSpaceCoursGit/coursgit (master)
$ ll
```

On peut également voir le contenu complet du projet

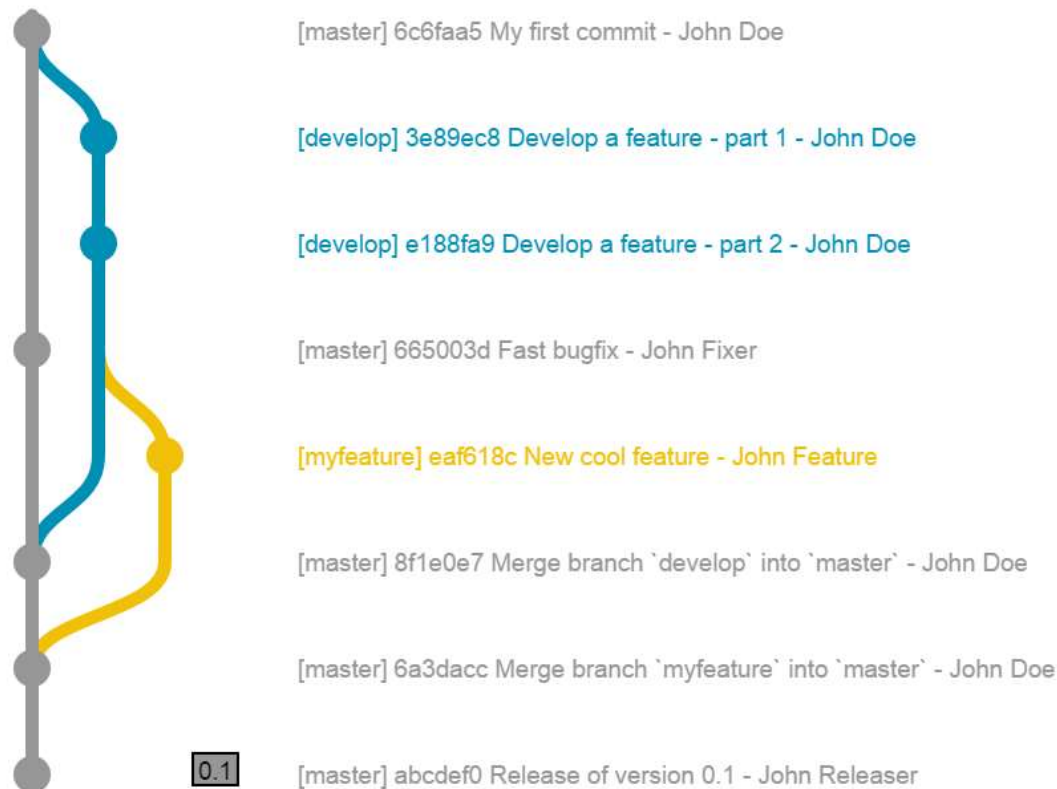
MINGW64:/d/WorkSpaceYaum/workSpaceCoursGit/coursgit

```
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/workSpaceYaum/workSpaceCoursGit/coursgit (master)
$ ll
total 2
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 archives/
-rw-r--r-- 1 Baron Eraser 197121 21 sept. 24 15:41 fichiersInutiles.txt
-rw-r--r-- 1 Baron Eraser 197121 48 sept. 24 15:41 lienOriginal.txt
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie1/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie2/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie3/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie4/
```

• C- Les branches

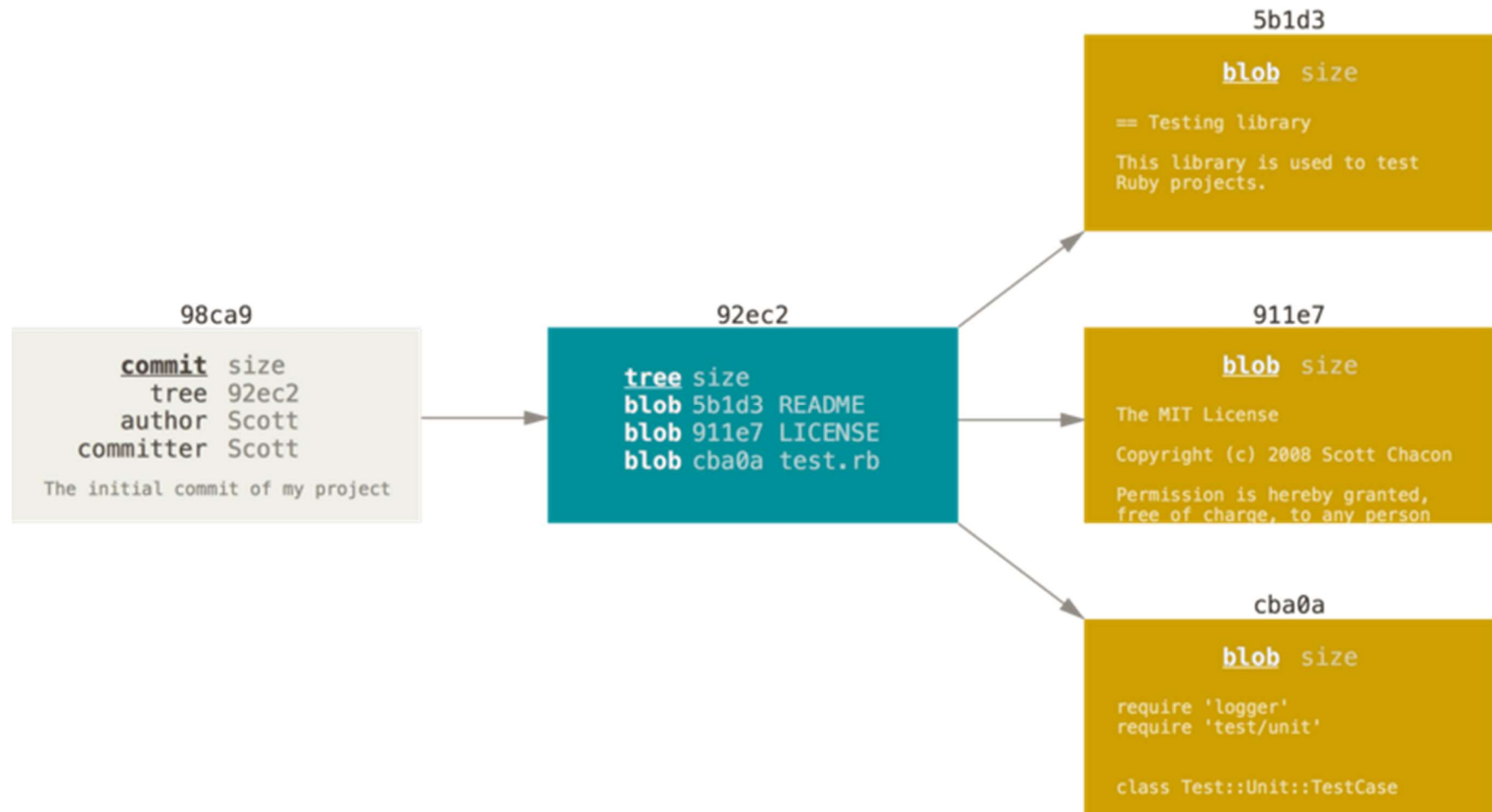
- git branch

Presque tous les VCS proposent une certaine forme de gestion de branches. Créer une branche signifie diverger de la ligne principale de développement et continuer à travailler sans impacter cette ligne.



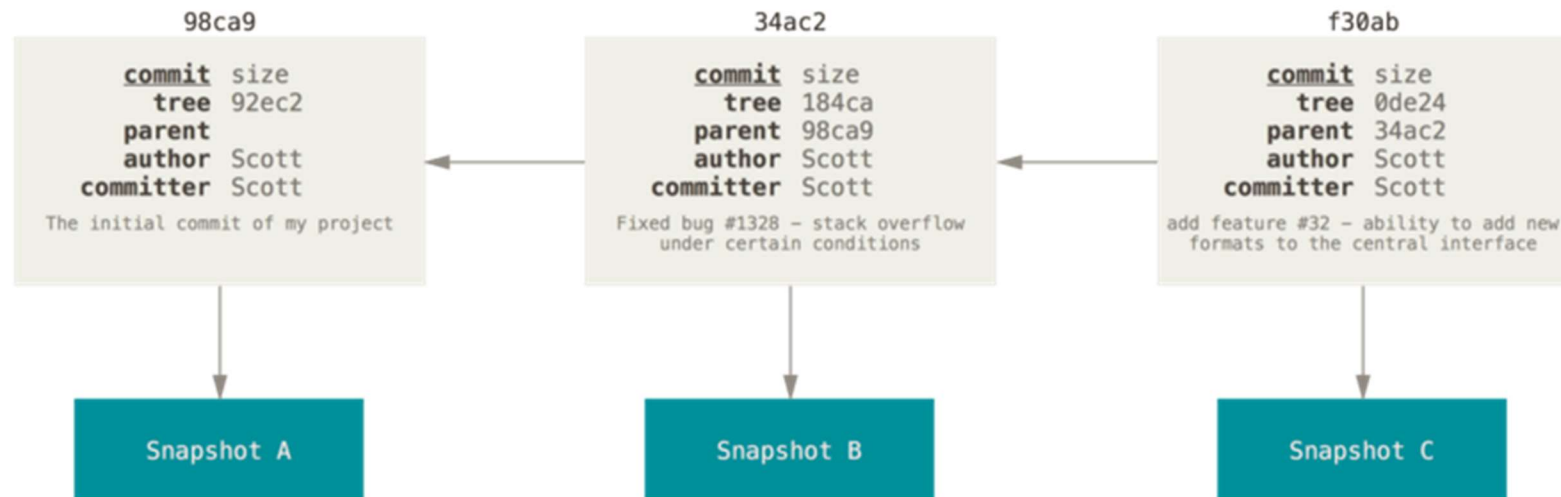


Lorsque vous faites un commit (c'est lorsque vous validez des modifications et que vous demandez à Git de les enregistrer), Git stocke un objet commit qui contient un pointeur vers l'instantané (snapshot) du contenu que vous avez indexé.



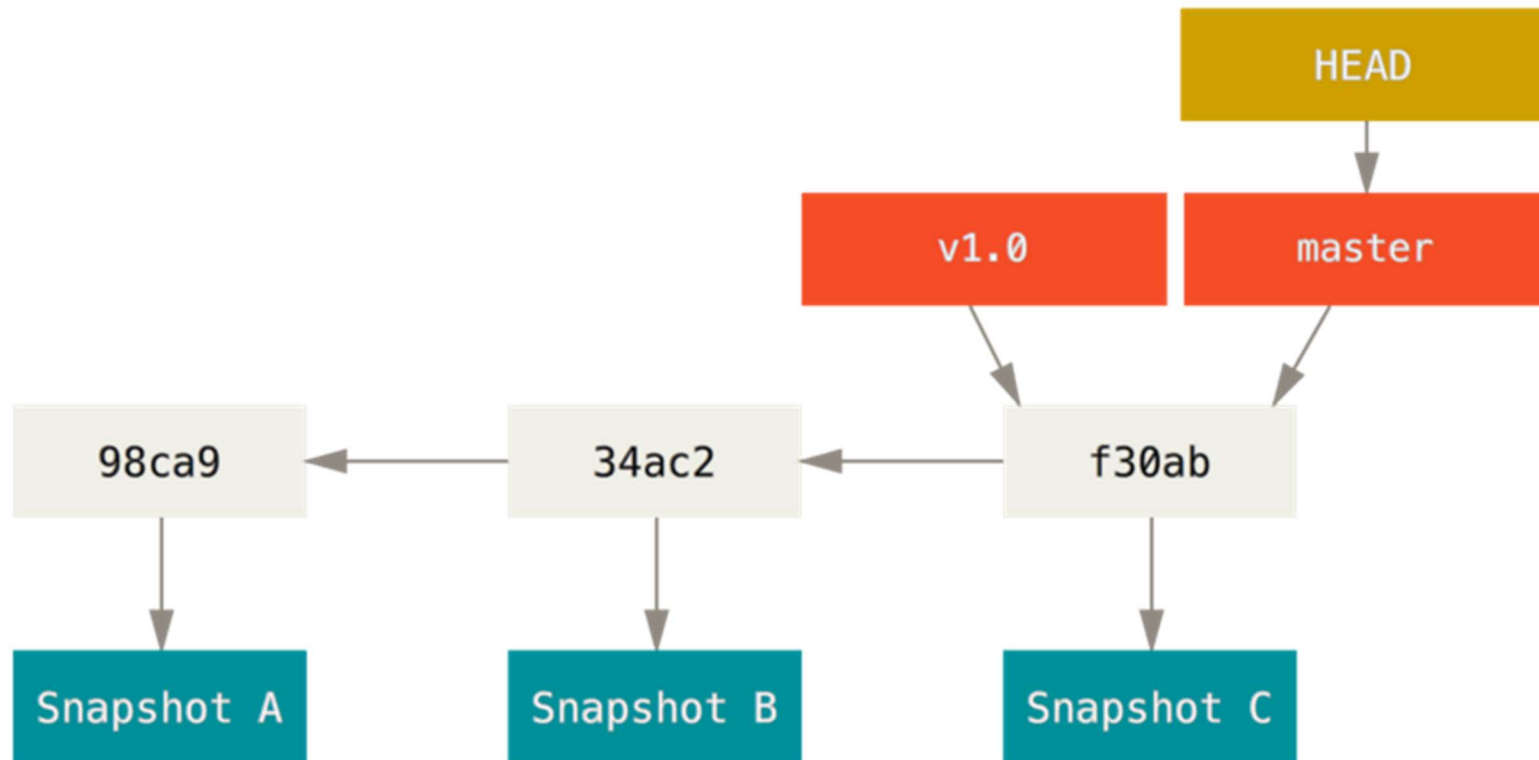
Un commit et son arbre

Si vous faites des modifications et validez à nouveau, le prochain commit stocke un pointeur vers le commit le précédant immédiatement.



Commits et leurs parents

Une branche dans Git est simplement un pointeur léger et déplaçable vers un de ces commits.



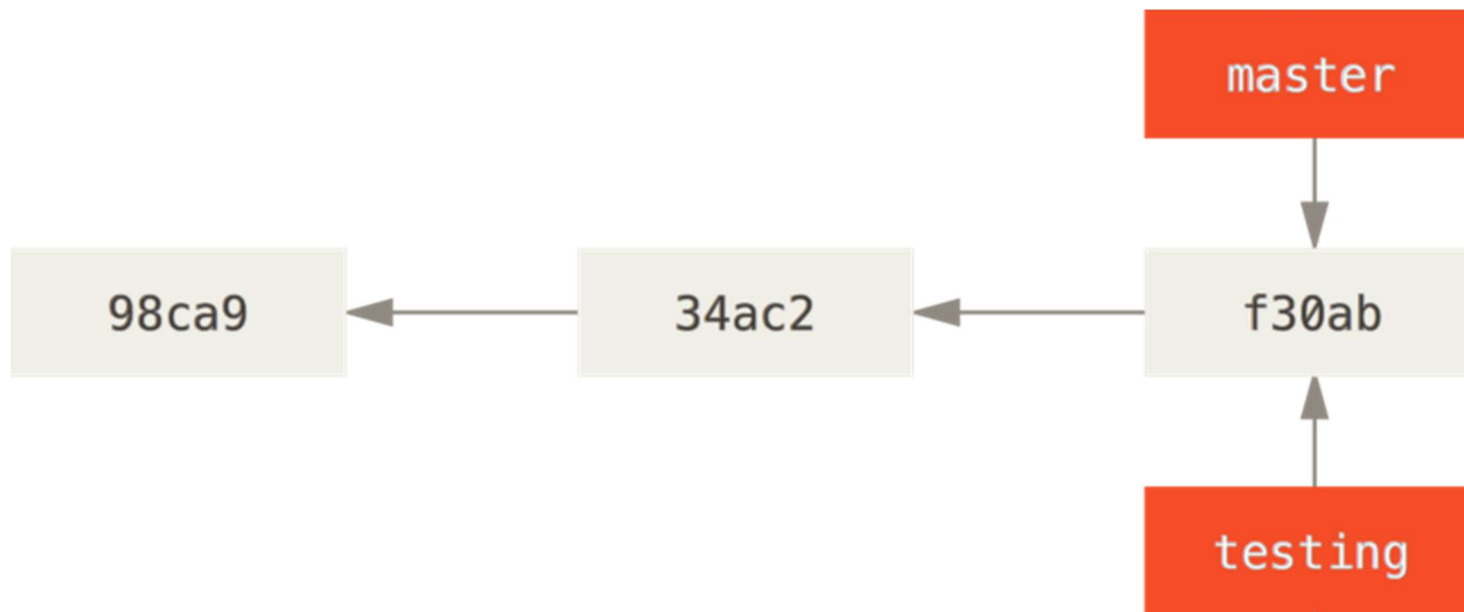
Une branche et l'historique de ses commits

- Créer une nouvelle branche

Que se passe-t-il si vous créez une nouvelle branche ? Eh bien, cela crée un nouveau pointeur pour vous. Supposons que vous créez une nouvelle branche nommée testing. Vous allez utiliser pour cela la commande git branch :

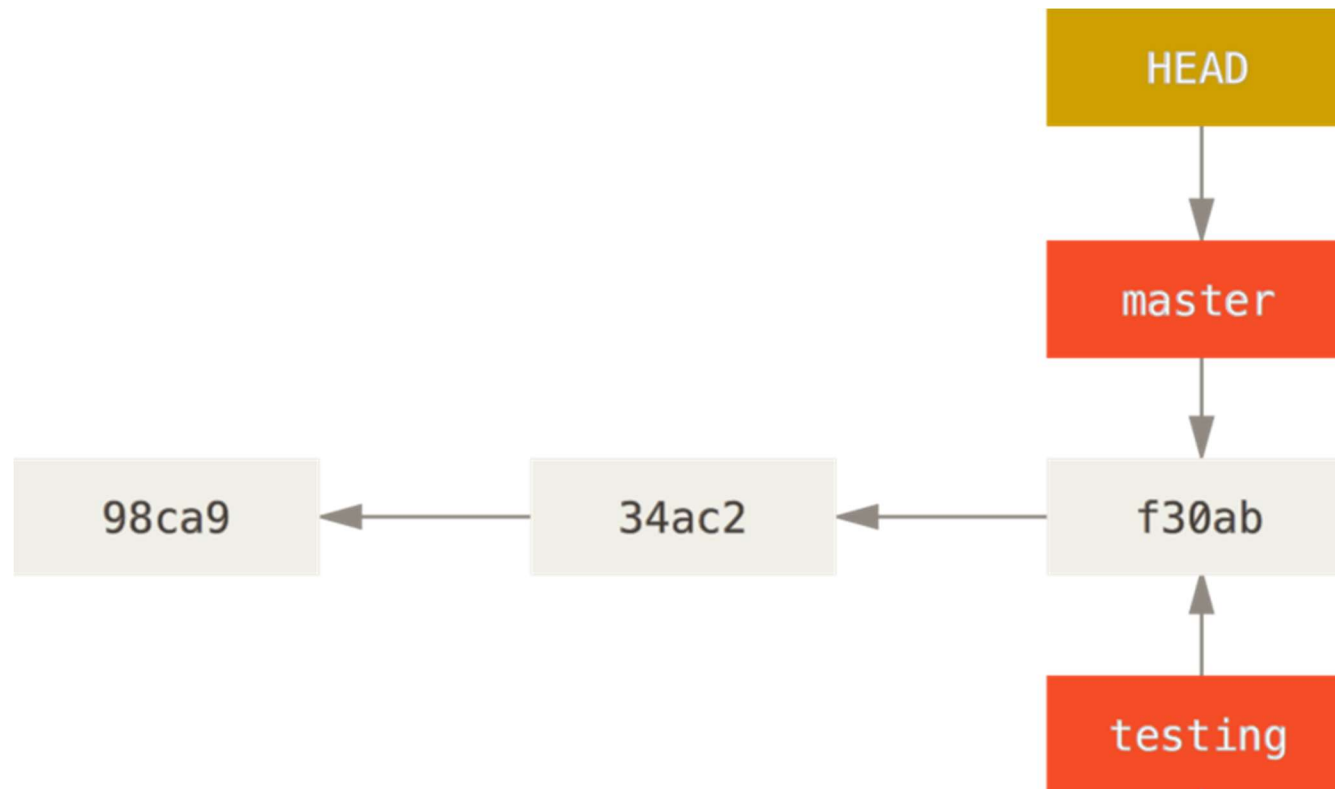
```
$ git branch testing
```

Cela crée un nouveau pointeur vers le commit courant.



Deux branches pointant vers la même série de commits

Comment Git connaît-il alors la branche sur laquelle vous vous trouvez ?



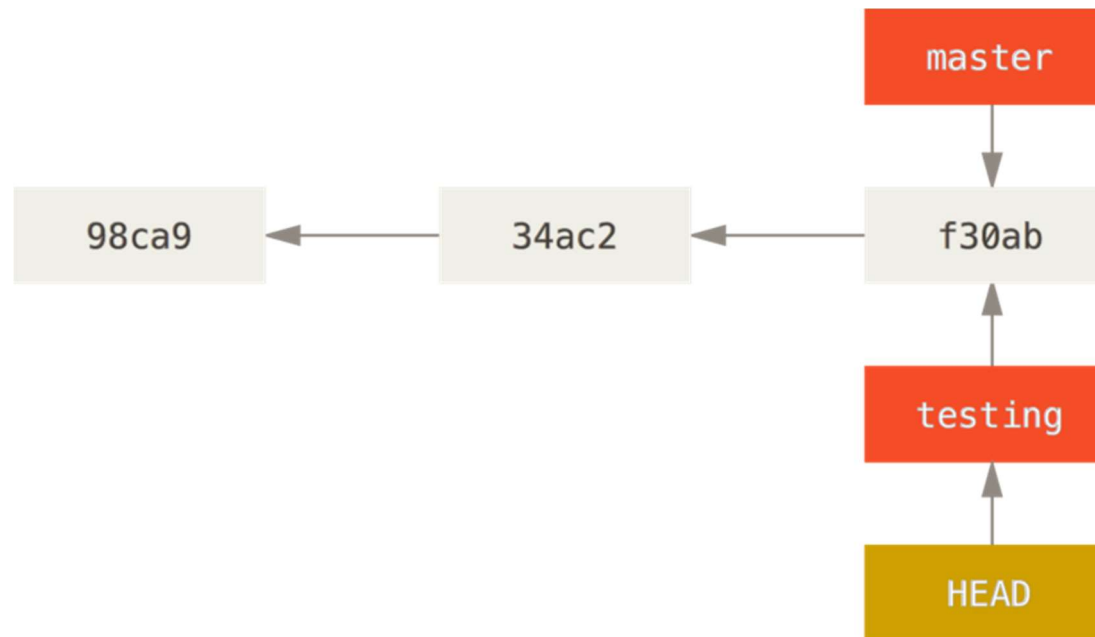
HEAD pointant vers une branche

- Basculer entre les branches

Pour basculer sur une branche existante, il suffit de lancer la commande git checkout. Basculons sur la nouvelle branche testing :

```
$ git checkout testing
```

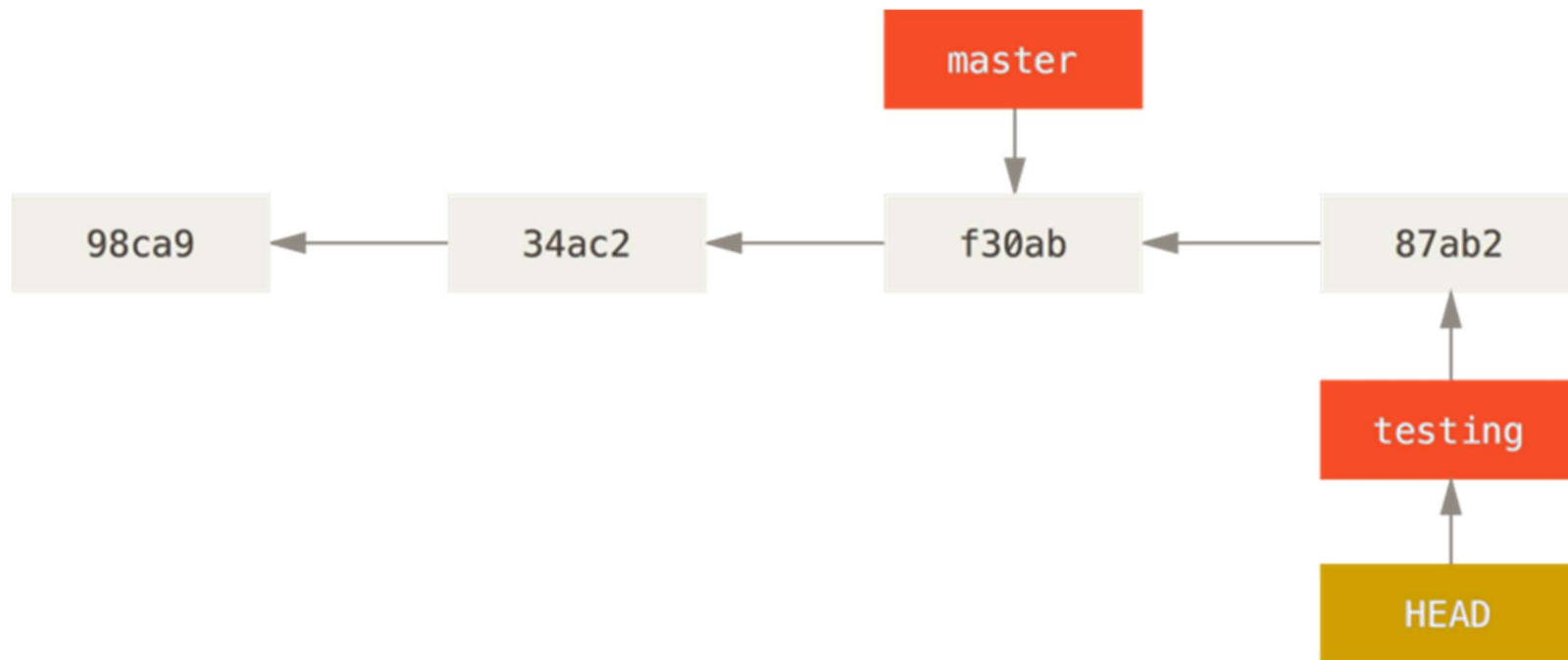
Cela déplace HEAD pour le faire pointer vers la branche testing.



HEAD pointe vers la branche courante

Qu'est-ce que cela signifie ? Et bien, faisons une modification dans le fichier test.rb que nous avons ajouté dans notre exemple et validons cette modification (ou commit) :

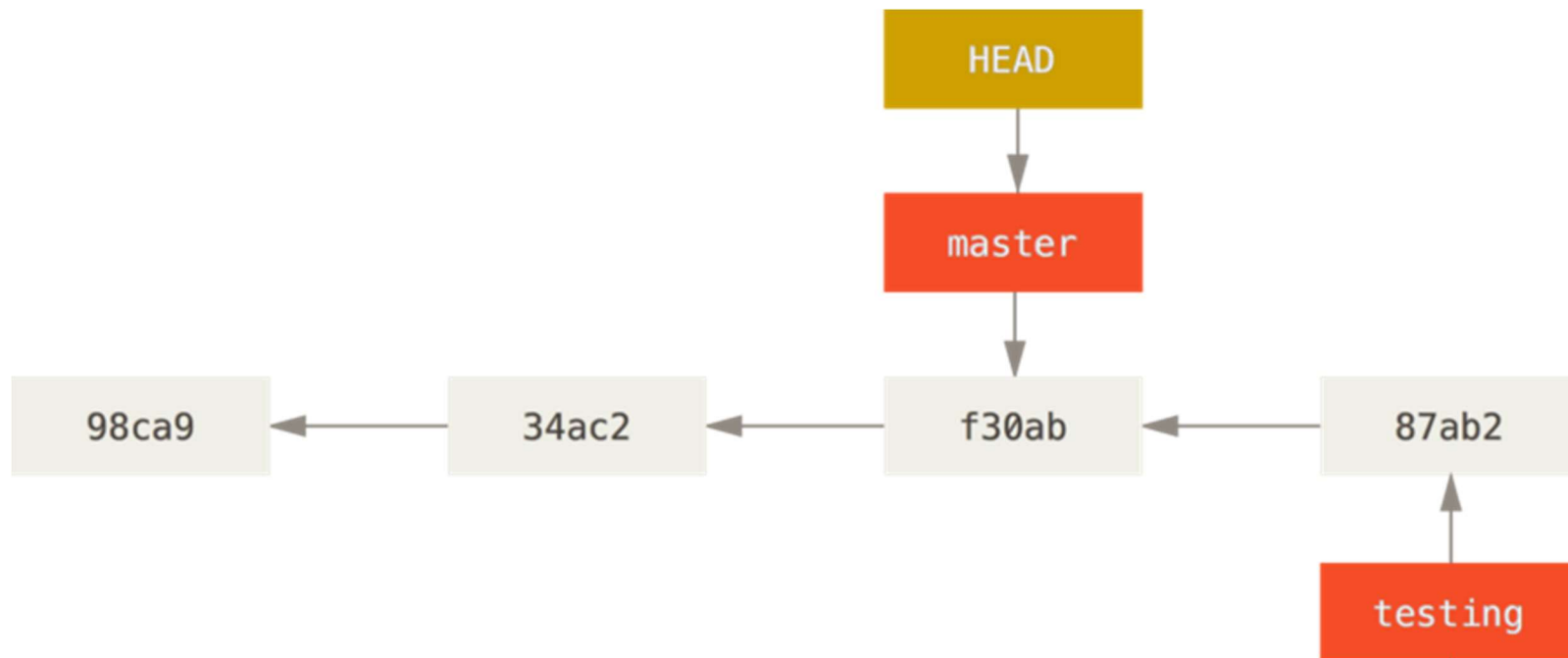
```
$ vim test.rb
$ git commit -a -m 'made a change'
```



Le pointeur HEAD avance à chaque commit

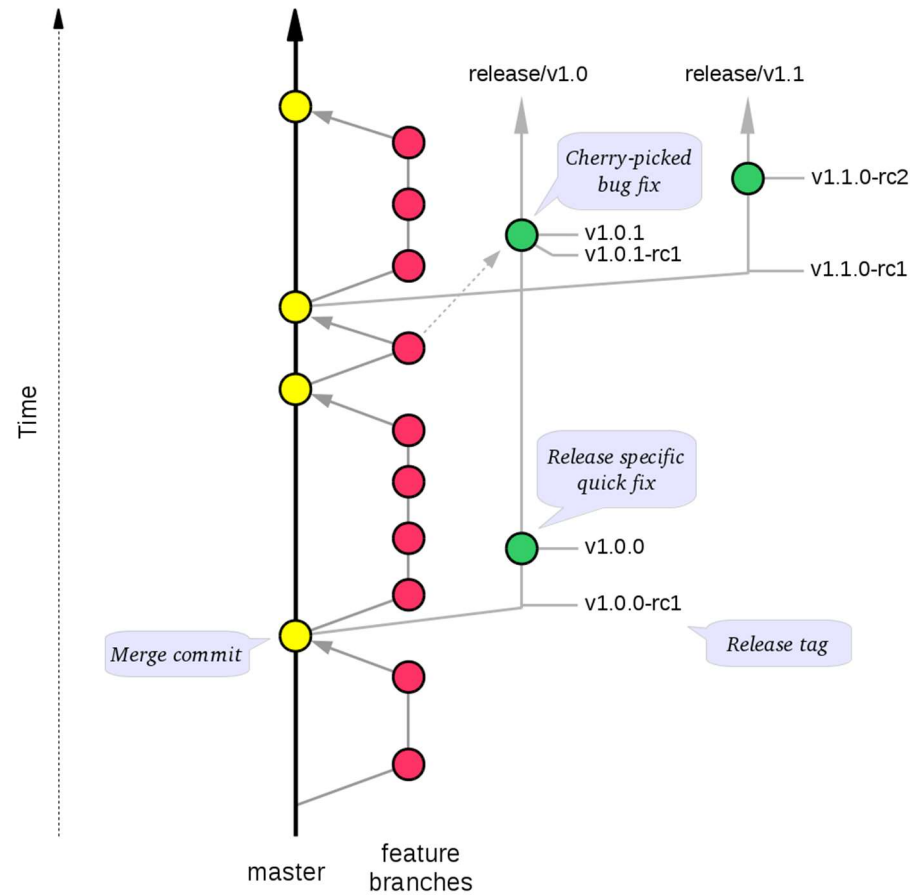
C'est intéressant parce qu'à présent, votre branche test a avancé tandis que la branche master pointe toujours sur le commit sur lequel vous étiez lorsque vous avez lancé la commande git checkout pour changer de branche. Retournons sur la branche master :

```
$ git checkout master
```



HEAD est déplacé lors d'un checkout

- Changer de branche modifie les fichiers dans votre répertoire de travail

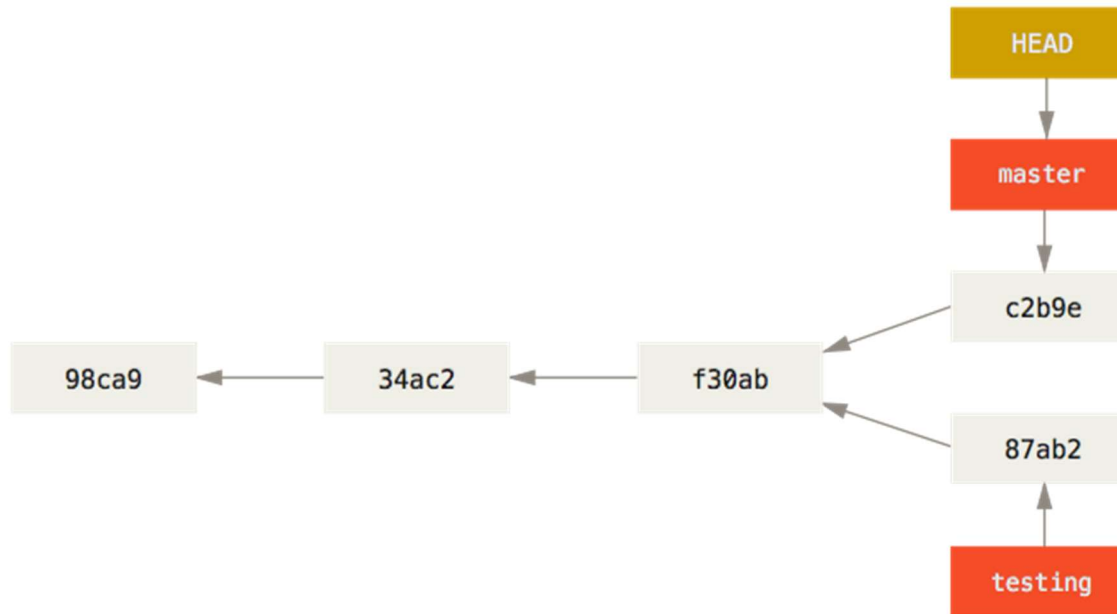


Réalisons quelques autres modifications et validons à nouveau :

```
$ vim test.rb
$ git commit -a -m 'made other changes'
```

Maintenant, l'historique du projet a divergé. Vous avez créé une branche et basculé dessus, y avez réalisé des modifications, puis vous avez rebasculé sur la branche principale et réalisé d'autres modifications.

Et vous avez fait tout ceci avec de simples commandes : `branch`, `checkout` et `commit`.



Divergence d'historique

- TD - git branch

Revenons dans notre console et regardons de plus près la barre d'adresse avec entre parenthèse le terme (master) :

MINGW64; /d/WorkSpaceYaum/workSpaceCoursGit/coursgit

```
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/workSpaceYaum/workSpaceCoursGit/coursgit (master)
```

Git nous informe que nous sommes sur la branche « master ».



**IL EST STRICTEMENT INTERDIT
DE TRAVAILLER SUR LA
BRANCHE MASTER !**



Il nous faut donc :

- Soit créer une nouvelle branche
- Soit changer de branche

Pour connaître s'il existe des branches, git a des commandes toutes prêtes :

Si vous faites : `git branch`

```
$ git branch
* master
```

Git nous informe que nous sommes sur la branche master, mais ne nous indique pas qu'il y en a d'autres. Pour connaître s'il existe d'autres branches, faisons :

```
$ git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/master
remotes/origin/qualif
```

Et là nous voyons qu'il en existe d'autre, mais en local, pas sur NOTRE repository local.

Pour cela vous allez faire :

```
$ git checkout qualif
```

Et vous devriez voir ceci :

```
$ git checkout qualif
Switched to a new branch 'qualif'
Branch 'qualif' set up to track remote branch 'qualif' from 'origin'.
```

Et si maintenant vous faites : git branch

```
$ git branch
  master
* qualif
```

On peut voir désormais la branche qualif en local.

Maintenant vous allez créer votre propre branche. Pour cela vous vous placez sur la branche master. Vous faites :

```
$ git branch devVotrePrenom
```

Moi c'était devYan.

Ensuite vous faites la commande :

```
$ git branch  
devYan  
* master  
qualif
```

Et vous devriez voir apparaître le nom de votre branche. Brancher vous dessus !

```
Baron Eraser@DESKTOP-KP65DS0 MINGW64 /d/workSpaceYaum/workSpaceCoursGit/coursgit (devYan)  
$ |
```


- **D- Le status**

- **git status**

Souvenez-vous que chaque fichier de votre copie de travail peut avoir deux états :

- sous suivi de version
- ou non suivi.

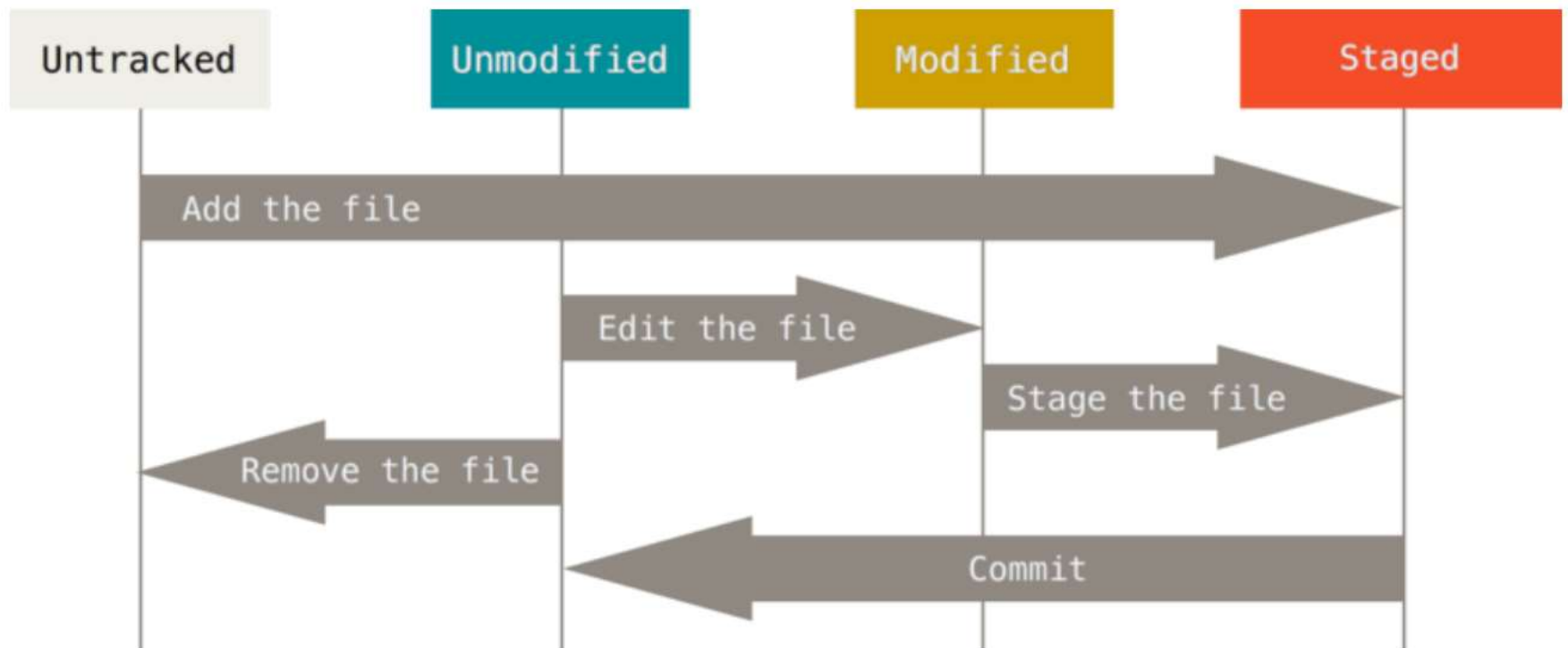
Les fichiers suivis sont les fichiers qui appartenaient déjà au dernier instantané ; ils peuvent être inchangés, modifiés ou indexés.



GIT alias shortcuts in terminal

```
alias gs='git status'  
alias ga='git add'  
alias gp='git push'  
alias c='git commit'
```

Voici le cycle de vie des fichiers sous Git :



Et l'outil principal pour déterminer quels fichiers sont dans quel état est la commande `git status`.

- TD - git status

Revenons à notre console et tapez la commande suivante :

```
$ git status  
On branch devYan  
nothing to commit, working tree clean
```

Ce message signifie que votre copie de travail est propre, en d'autres mots, aucun fichier suivi n'a été modifié.

Supposons que vous ajoutiez un nouveau fichier à votre projet, un simple fichier texte que vous allez appeler NomPrenom.txt.

Moi je vais passer par l'éditeur VIM, mais vous pouvez prendre n'importe quel éditeur. Il faut que le fichier soit dans votre dossier racine de votre projet. Après l'avoir créé faite ll :

```
$ ll
total 3
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 archives/
-rw-r--r-- 1 Baron Eraser 197121 24 sept. 27 09:49 aumagyYannick.txt
-rw-r--r-- 1 Baron Eraser 197121 21 sept. 24 15:41 fichiersInutiles.txt
-rw-r--r-- 1 Baron Eraser 197121 48 sept. 24 15:41 lienOriginal.txt
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie1/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie2/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie3/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie4/
```

On doit voir apparaître le nouveau fichier dans la liste.

Si l'on veut trier par date : ll -ltr :

```
$ ll -ltr
total 3
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 archives/
-rw-r--r-- 1 Baron Eraser 197121 48 sept. 24 15:41 lienOriginal.txt
-rw-r--r-- 1 Baron Eraser 197121 21 sept. 24 15:41 fichiersInutiles.txt
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie1/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie2/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie3/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie4/
-rw-r--r-- 1 Baron Eraser 197121 24 sept. 27 09:49 aumagyYannick.txt
```

Si ce fichier n'existait pas auparavant, ce qui doit être le cas, et que vous lancez la commande `git status`, vous verrez votre fichier non suivi comme ceci :

```
$ git status
On branch devYan
Untracked files:
  (use "git add <file>..." to include in what will be committed)

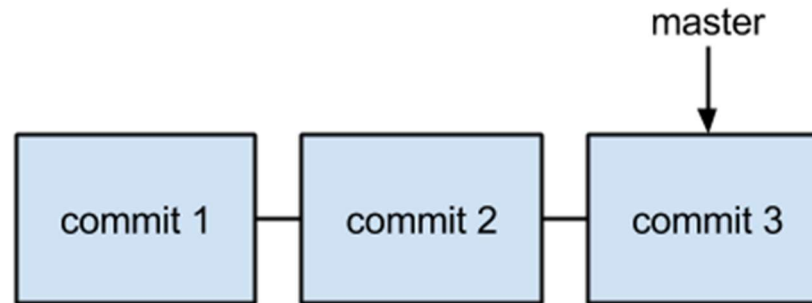
    aumagyYannick.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Vous pouvez constater que votre nouveau fichier **nomPrenom.txt** n'est pas en suivi de version, car il apparaît dans la section « Untracked files » de l'état de la copie de travail. « Untracked » signifie simplement que Git détecte un fichier qui n'était pas présent dans le dernier instantané ; Git ne le placera sous suivi de version que quand vous lui indiquerez de le faire.

- **E- Les commits**

- **git commit**



Que signifie « commit » pour Git ? Il provient du latin *committere*, de « *co(m)* », signifiant « ensemble » et « *mittere* », signifiant « envoyer ». Il conserve en anglais plusieurs sens, celui de confier (comme on confie une mission) et d'effectuer une action. Dans Git, il s'agit de valider une action.

Dans ce contexte, la validation est souvent accompagnée d'un commentaire décrivant le contenu de la modification apportée. Ce commentaire est souvent structuré ainsi : {date - login - commentaire}

- TD - git commit

Bien, retournons dans notre console. Vérifiez que vous êtes bien sur votre branche.

Tapez la commande suivante :

```
$ git commit
On branch devYan
Untracked files:
  aumagyYannick.txt

nothing added to commit but untracked files present
```

On peut voir que Git nous informe qu'un fichier est untracked. Comme nous l'avons vu c'est le premier stade du cycle de vie des états de fichier sous Git. Pour commencer à suivre un nouveau fichier, nous allons utiliser la commande git add.

Pour commencer à suivre le fichier nomPrenom.txt, tapez la commande suivante :

```
$ git add aumagyYannick.txt
warning: LF will be replaced by CRLF in aumagyYannick.txt.
The file will have its original line endings in your working directory
```


Si vous lancez à nouveau la commande `git status`, vous pouvez constater que votre fichier **nomPrenom.txt** est maintenant suivi et indexé :

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   aumagyYannick.txt
```

Maintenant, modifions un fichier qui est déjà sous suivi de version. Si vous modifiez le fichier sous suivi de version appelé **fichiersInutiles.txt** en ajoutant dedans une phrase de votre choix (totalitoto va à la plage sans son parasol) et que vous lancez à nouveau votre commande `git status`, vous verrez ceci :

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   aumagyYannick.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   fichiersInutiles.txt
```

On peut voir deux section d'états : « Changes to be committed : » et « Changes not staged for commit: ».

Le fichier **nomPrenom.txt** apparaît sous la section nommée « Changes not staged for commit » ce qui signifie que le fichier sous suivi de version a été modifié dans la copie de travail mais n'est pas encore indexé.

Pour l'indexer, il faut lancer la commande `git add`

Note : `git add` est une commande multi-usage. Elle peut être utilisée pour :

- placer un fichier sous suivi de version,
- pour indexer un fichier
- pour marquer comme résolus des conflits de fusion de fichiers.

Lançons maintenant `git add` pour indexer le fichier **fichierInutiles.txt**, et relançons la commande `git status` :

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   aumagyYannick.txt
    modified:   fichiersInutiles.txt
```

À présent, les deux fichiers sont indexés et feront partie de la prochaine validation. Mais supposons que vous souhaitiez apporter encore une petite modification au fichier **fichiersInutiles.txt** avant de réellement valider la nouvelle version. Vous l'ouvrez à nouveau, réalisez la petite modification et vous voilà prêt à valider.

Néanmoins, nous allons lancer `git status` pour vérifier :

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   aumagyYannick.txt
    modified:   fichiersInutiles.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   fichiersInutiles.txt
```

Que s'est-il donc passé ? À présent, **fichiersInutiles.txt** apparaît à la fois comme indexé et non indexé.

En fait, Git indexe un fichier dans son état au moment où la commande git add est lancée.

Si le fichier est modifié après un git add, il faut relancer git add pour prendre en compte l'état actuel de la copie de travail :

```
$ git add fichiersInutiles.txt
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   aumagyYannick.txt
    modified:   fichiersInutiles.txt
```

Il nous reste maintenant à valider nos modifications.

Maintenant que votre zone d'index est dans l'état désiré, vous pouvez valider vos modifications.

Dans notre cas, la dernière fois que vous avez lancé `git status`, vous avez vérifié que tout était indexé, et vous êtes donc prêt à valider vos modifications. La manière la plus simple de valider est de taper `git commit` :

```
$ git commit
```

Cela va ouvrir automatiquement l'éditeur de texte de la console. Il faut absolument mettre un commentaire, cela fait partie des « bonnes pratiques » de travail.

Tapez : `i` (pour insertion), puis entrez ce texte :

```
Mon premier commit
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch devYan
# Changes to be committed:
#   new file:   aumagyYannick.txt
#   modified:   fichiersInutiles.txt
#
```

Puis faite « `echap` », puis tapez : suivi de `wq`

Vous devriez voir apparaître ceci :

```
$ git commit
[devYan 4a0a508] Mon premier commit
 2 files changed, 4 insertions(+), 1 deletion(-)
 create mode 100644 aumagyYannick.txt
```

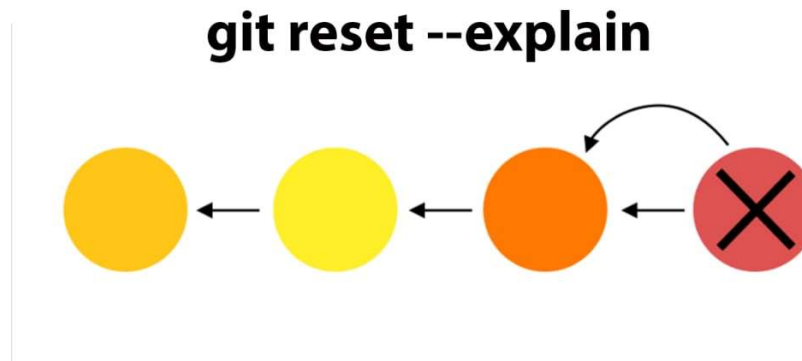
Bravo 😊, à présent, vous avez créé votre premier commit ! Vous pouvez constater que le commit vous fournit quelques informations sur lui-même :

- sur quelle branche vous avez validé (devVotrePrenom),
- quelle est sa somme de contrôle SHA-1 (463dc4f),
- combien de fichiers ont été modifiés,
- et quelques statistiques sur les lignes ajoutées et effacées dans ce commit.

• F- Annuler des actions

- git reset

La commande git reset sert à désindexer un fichier déjà indexé. Nous allons voir comment faire des modifications dans votre zone d'index et votre zone de travail.



- TD - reset

Maintenant que nous avons modifié nos deux fichiers, je souhaite les valider comme deux modifications indépendantes. Mais que vous avez tapé accidentellement git add * et donc indexé les deux. Comment annuler l'indexation d'un des fichiers car par exemple nous estimons qu'il n'est pas terminé ?

La commande git status vous le rappelle.

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   aumagyYannick.txt
        modified:   fichiersInutiles.txt
```

Pour dés-indexer le fichier nomPrénom.txt, il suffit de faire la commande suivante :

```
$ git reset HEAD aumagyYannick.txt
Unstaged changes after reset:
M      aumagyYannick.txt
```

Confirmons par git status :

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   fichiersInutiles.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   aumagyYannick.txt
```

Ce qui signifie que si nous commitons, seul le fichier « fichiersInutiles.txt » sera validé.

- **git amend**



C'est donc un des rares cas d'utilisation de Git où des erreurs de manipulation peuvent entraîner des pertes définitives de données.

Si vous souhaitez rectifier cette erreur, vous pouvez valider le complément de modification avec l'option `–amend`.

- **TD - git amend**

Tapez la commande suivante

```
$ git commit --amend
```

Cela va ouvrir l'éditeur de la console. Nous pouvons voir que Git ouvre le dernier commit en ajoutant le fichier indexé précédemment. Dans le texte en haut ajoutez le mot modifié à la fin puis enregistrez et fermez.

Tapez la commande suivante

```
$ git status
On branch devYan
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   aumagyYannick.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Qu'a fait la commande `–amend` ? Celle-ci prend en compte la zone d'index et l'utilise pour le commit.

Nous avons vu que l'éditeur de message de validation a démarré, et qu'il contenait déjà le message de la validation précédente.

- **git checkout**

Que faire si vous réalisez que vous ne souhaitez pas conserver les modifications du fichier **nomPrenom.txt** ?

Heureusement, git status vous indique comment procéder. Dans le résultat de la dernière commande, la zone de travail ressemble à ceci :

```
$ git status
On branch devYan
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   aumagyYannick.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

- **TD - git checkout**

Ce qui vous indique de façon explicite comment annuler des modifications que nous avons faites.

Ouvrez votre fichier nomPrenom.txt et regardez son contenu ajoutez le texte suivant : ceci est « ma modification ultime » puis fermez le.

Entrons la commande suivante pour annuler les dernières modifications.

```
$ git checkout - nomPrenom.txt
```

```
$ git status  
On branch devYan  
nothing to commit, working tree clean
```

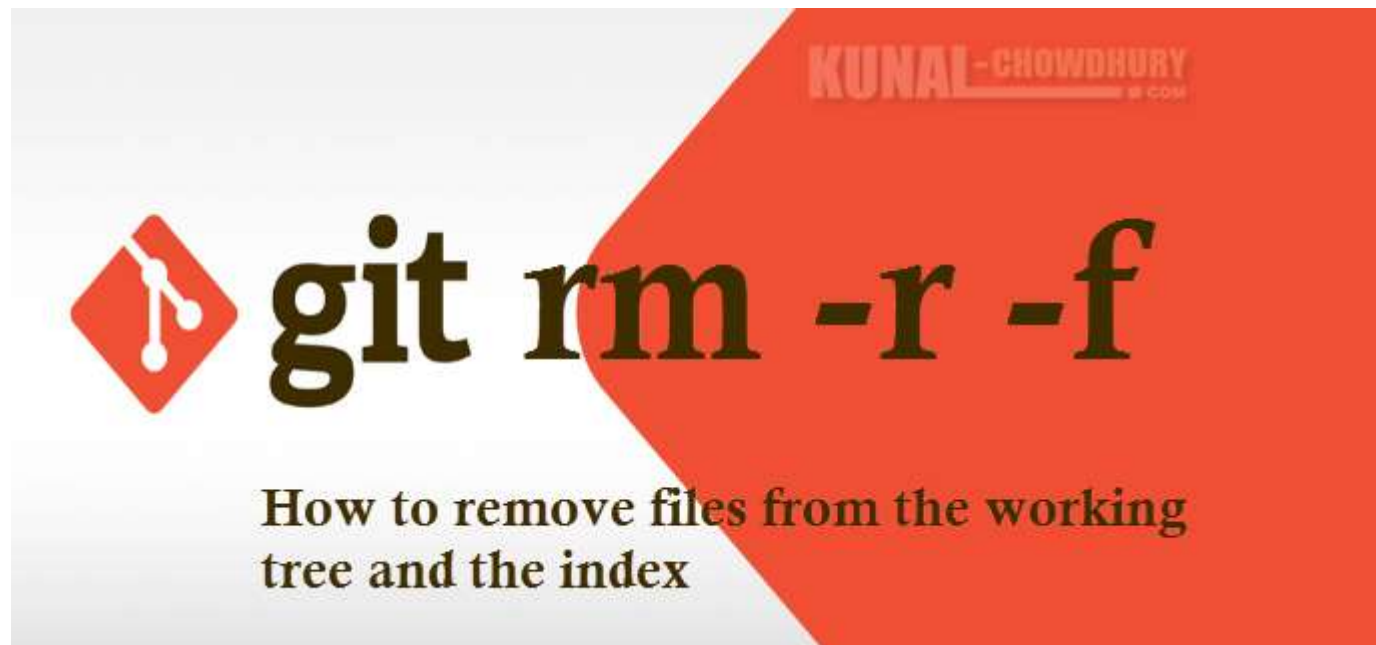
Git status nous dis que tout est ok pour lui.

Ouvrez de nouveau votre fichier nomPrenom.txt et regardez son contenu : il est revenu à son état initial en annulant TOUTES LES MODIFICATIONS QUE NOUS N'AVONS PAS INDEXEES.

- **G- Supprimer des fichiers et des branches**

- **git rm**

Pour effacer un fichier de Git, vous devez l'éliminer des fichiers en suivi de version (plus précisément, l'effacer dans la zone d'index) puis valider.



- TD – git rm

Tout d’abord, commençons par créer un fichier que nous allons ensuite supprimer (oui c’est idiot 😊).

Tapez la commande :

```
$ vim fichierASupprimer.txt
```

Taper un texte a l’intérieur. Fermer vim. Tapez la commande :

```
$ ll
total 4
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 archives/
-rw-r--r-- 1 Baron Eraser 197121 25 sept. 27 11:25 aumagyYannick.txt
-rw-r--r-- 1 Baron Eraser 197121 7 sept. 27 12:52 fichierASupprimer.txt
-rw-r--r-- 1 Baron Eraser 197121 93 sept. 27 11:12 fichiersInutiles.txt
-rw-r--r-- 1 Baron Eraser 197121 48 sept. 24 15:41 lienOriginal.txt
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie1/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie2/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie3/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie4/
```

Qui devrait vous afficher ceci ci-dessus.

Maintenant ajoutez a l'index le fichier créé. Faites un git status.

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   fichierASupprimer.txt
```

Pour supprimer le fichier : rm **fichierASupprimer.txt**

Si nous faisons un rm du fichier maintenant, git le détecte et n'est pas content :

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   fichierASupprimer.txt

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    deleted:    fichierASupprimer.txt
```

Du coup nous devons le supprimer de l'index par :

```
$ git rm fichierASupprimer.txt  
rm 'fichierASupprimer.txt'
```

Maintenant notre branche est ok.

```
$ git status  
On branch devYan  
nothing to commit, working tree clean
```

Recréons notre fichier et ajoutons-le à l'index.

```
$ vim fichierASupprimer.txt
```

Taper un texte a l'intérieur. Fermer vim. Vérifier par ll qu'il y est bien. Ajoutez-le à l'index.

Vérifions :

Le statut de git :

```
$ git status
On branch devYan
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   fichierASupprimer.txt
```

La présence de notre fichier :

```
$ ll
total 4
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 archives/
-rw-r--r-- 1 Baron Eraser 197121 25 sept. 27 11:25 aumagyYannick.txt
-rw-r--r-- 1 Baron Eraser 197121 6 sept. 27 13:02 fichierASupprimer.txt
-rw-r--r-- 1 Baron Eraser 197121 93 sept. 27 11:12 fichiersInutiles.txt
-rw-r--r-- 1 Baron Eraser 197121 48 sept. 24 15:41 lienOriginal.txt
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie1/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie2/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie3/
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie4/
```

Tapons maintenant :

```
$ git rm -f fichierASupprimer.txt  
rm 'fichierASupprimer.txt'
```

Note il se peut que git détecte une erreur. Ajoutez l'option -f :

```
$ git rm -f fichierASupprimer.txt  
rm 'fichierASupprimer.txt'
```

Vérifions le statut de git :

```
$ git status  
On branch devYan  
nothing to commit, working tree clean
```

Et la non présence de notre fichier :

```
$ ll  
total 3  
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 archives/  
-rw-r--r-- 1 Baron Eraser 197121 25 sept. 27 11:25 aumagyYannick.txt  
-rw-r--r-- 1 Baron Eraser 197121 93 sept. 27 11:12 fichiersInutiles.txt  
-rw-r--r-- 1 Baron Eraser 197121 48 sept. 24 15:41 lienOriginal.txt  
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie1/  
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie2/  
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie3/  
drwxr-xr-x 1 Baron Eraser 197121 0 sept. 24 15:41 partie4/
```

- **git branch -d**

Cette commande permet de supprimer des branches sur le repository local.

- **TD – git branch -d**

- Créer une nouvelle branche devPrenomASupprimer
- Switchez dessus.
- Tapez ll. Vous voyez que c'est une copie de votre branche devPrenom.
- Switchez sur votre branche devPrenom
- Listez les branches
- Supprimez la branche devPrenomASupprimer
- Listez les branches

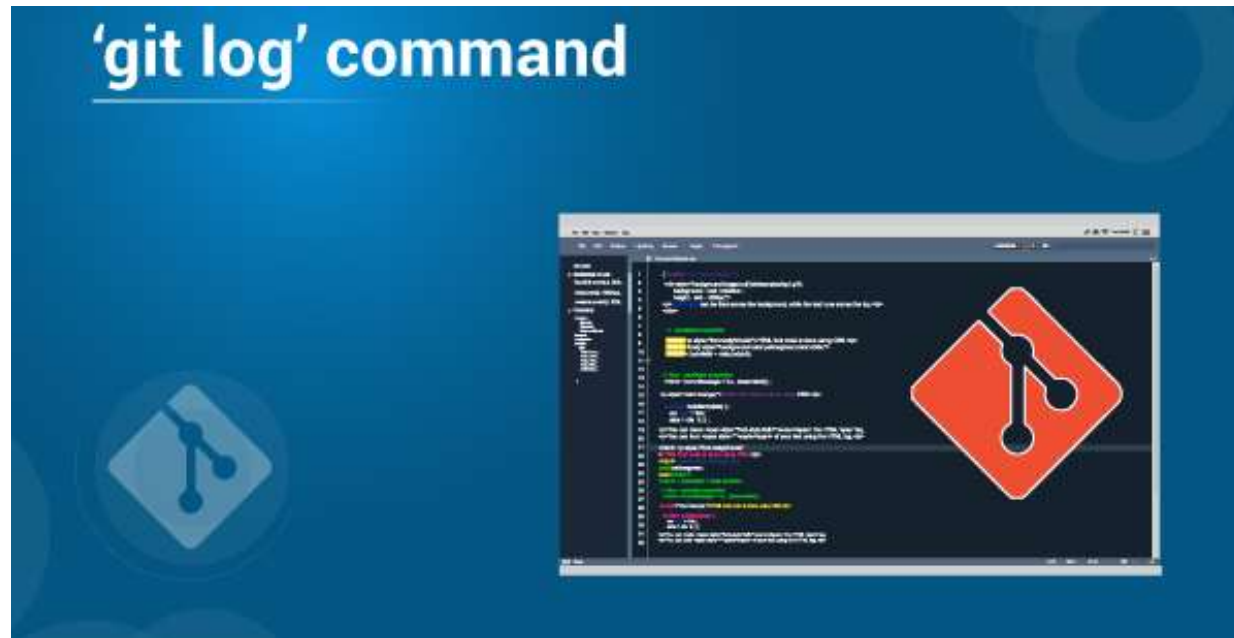
```
$ git branch
* devYan
  master
  qualif
```

devPrenomASupprimer à disparue 😊

• H- Consulter l'historique

- git log

L'intérêt d'utiliser Git est de pouvoir consulter n'importe quelle version antérieure du projet : son historique.



Attention cependant :

- à ne pas avoir de modification non commitée ;
- à ne pas faire de modification sur une version ancienne.

- TD - git log

Revenons à notre console et tapez :

```
$ git log
commit 8db419ad5419cfb21da653ddc7eb0449d4931ff3 (HEAD -> devYan)
Author: micheligestpro <tech@igestpro.com>
Date:   Fri Sep 27 10:23:43 2019 +0200

    Mon premier commit modifié

commit b5e17b02fb22bd8752a991b0673b405874517aa6 (origin/qualif, origin/master,
origin/HEAD, qualif, master)
Author: micheligestpro <tech@igestpro.com>
Date:   Tue Sep 24 15:38:26 2019 +0200

    Initial commit
```

Et cela devrait nous donner quelque chose comme cela.

En ajoutant l'option -p, vous visualiserez les différences dans le contenu du fichier entre chaque commit. Appuyer que Q pour quitter 😊

Par défaut, git log invoqué sans argument énumère en ordre chronologique inversé les commits réalisés. Cela signifie que les commits les plus récents apparaissent en premier.

git log dispose d'un très grand nombre d'options permettant de paramétrer exactement ce que l'on cherche à voir. Nous allons détailler quelques-unes des plus utilisées.

```
$ git log -p -1
commit 8db419ad5419cfb21da653ddc7eb0449d4931ff3 (HEAD -> devYan)
Author: micheligestpro <tech@igestpro.com>
Date:   Fri Sep 27 10:23:43 2019 +0200

    Mon premier commit modifié
(...)

```

Cette option affiche la même information mais avec un diff suivant directement chaque entrée.

Par exemple, si vous souhaitez visualiser des statistiques résumées pour chaque commit, vous pouvez utiliser l'option `--stat` :

```
$ git log --stat
commit 8db419ad5419cfb21da653ddc7eb0449d4931ff3 (HEAD -> devYan)
Author: micheligestpro <tech@igestpro.com>
Date:   Fri Sep 27 10:23:43 2019 +0200

    Mon premier commit modifié

    aumagyYannick.txt      | 1 +
    fichiersInutiles.txt  | 5 ++++
    2 files changed, 5 insertions(+), 1 deletion(-)

commit b5e17b02fb22bd8752a991b0673b405874517aa6 (origin/qualif, origin/master, origin/HEAD, qualif, master)
Author: micheligestpro <tech@igestpro.com>
Date:   Tue Sep 24 15:38:26 2019 +0200

    Initial commit

archives/99-toutLeLivre.docx | Bin 0 -> 876674 bytes
fichiersInutiles.txt         | 1 +
lienOriginal.txt             | 1 +
partie1/01-par_lesroutes.docx | Bin 0 -> 13267 bytes
partie1/02-les_uniformes.docx | Bin 0 -> 12900 bytes
partie1/03-les_dix.docx      | Bin 0 -> 12953 bytes
partie1/04-d_abord.docx      | Bin 0 -> 13023 bytes
partie1/fichiersInutiles1.txt | 1 +
partie2/05-de_nouveaux.docx  | Bin 0 -> 13650 bytes
partie2/06-hericourt.docx    | Bin 0 -> 13563 bytes
partie2/07-or_il_apercut.docx | Bin 0 -> 14835 bytes
partie2/08-riant_aux_eclat.docx | Bin 0 -> 14033 bytes
partie2/fichiersInutiles2.txt | 1 +
partie3/09-la_rage.docx      | Bin 0 -> 13657 bytes
partie3/10-il_mangeat.docx   | Bin 0 -> 13354 bytes
partie3/11-son_imaginbation.docx | Bin 0 -> 14033 bytes
partie3/12-groupe_silencieux.docx | Bin 0 -> 13517 bytes
partie3/fichiersInutiles3.txt | 1 +
partie4/13-sans_finir.docx   | Bin 0 -> 13793 bytes
partie4/14-bernard_compta.docx | Bin 0 -> 13894 bytes
partie4/15-la_conscience.docx | Bin 0 -> 14130 bytes
partie4/16-il_se_trouva.docx | Bin 0 -> 14261 bytes
partie4/fichiersInutiles4.txt | 1 +
23 files changed, 6 insertions(+)
```

Il existe pléthore de commande pour l'action git log. Voici un petit tableau récapitulatif :

Option	Description
<code>-p</code>	Affiche le patch appliqué par chaque commit
<code>--stat</code>	Affiche les statistiques de chaque fichier pour chaque commit
<code>--shortstat</code>	N'affiche que les ligne modifiées/insérées/effacées de l'option <code>--stat</code>
<code>--name-only</code>	Affiche la liste des fichiers modifiés après les informations du commit
<code>--name-status</code>	Affiche la liste des fichiers affectés accompagnés des informations d'ajout/modification/suppression
<code>--abbrev-commit</code>	N'affiche que les premiers caractères de la somme de contrôle SHA-1
<code>--relative-date</code>	Affiche la date en format relatif (par exemple "2 weeks ago" : il y a deux semaines) au lieu du format de date complet
<code>--graph</code>	Affiche en caractères ASCII le graphe de branches et fusions en vis-à-vis de l'historique
<code>--pretty</code>	Affiche les <i>commits</i> dans un format alternatif. Les formats incluent <code>oneline</code> , <code>short</code> , <code>full</code> , <code>fuller</code> , et <code>format</code> (où on peut spécifier son propre format)
<code>--oneline</code>	Option de convenance correspondant à <code>--pretty=oneline --abbrev-commit</code>



• **Quizz 3 – 60 mn**

1-

2-

3-

4-

5-

6-

7-