

Formation Git

IV - Historique et Dépôt

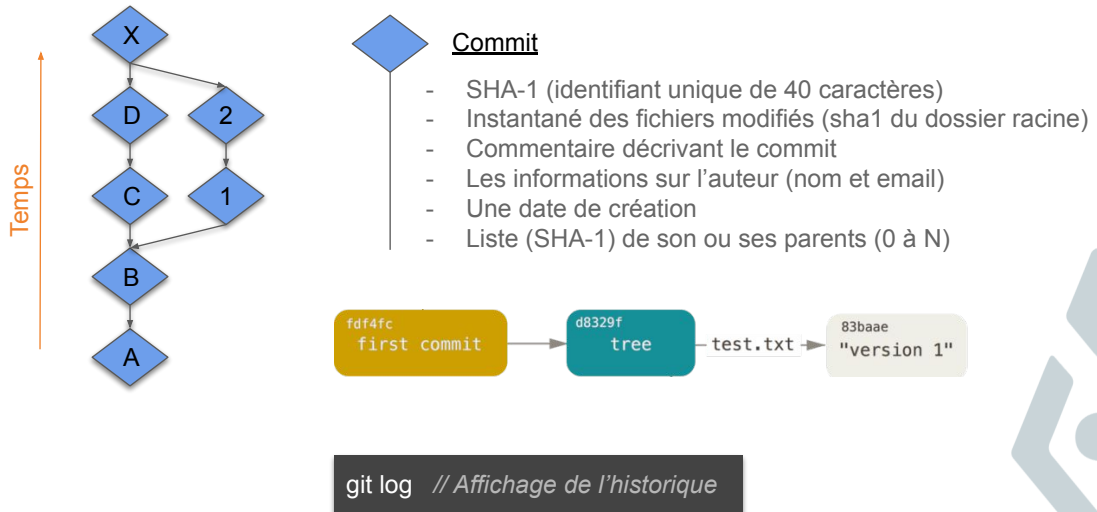


Arnaud MERCIER
arnaud.mercier@hexotech.fr



Nous allons voir comment visualiser et utiliser notre historique de commits via Git

Commits et historique



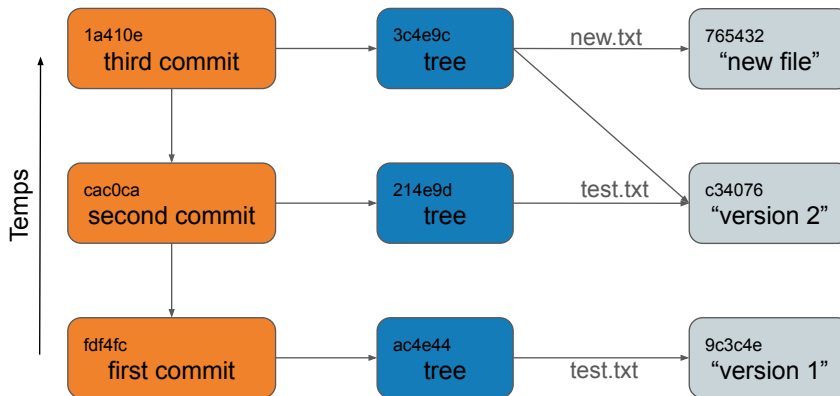
Un commit est constitué d'un ensemble d'informations:

- le sha1: identifiant unique de 40 caractères qui est également le checksum du commit
- l'instantané de la zone d'index au moment du commit
- Commentaire décrivant le commit
- Les informations sur l'auteur (nom et email)
- Une date de création
- Liste (SHA-1) de son ou ses parents (0 à N)

EXEMPLE (GOT):

1. `git log`

Commits et historique



Les objets se trouvent dans `.git/objects/`

Git enregistre uniquement les instantanés des fichiers modifiés.

Dans ce schéma nous pouvons voir qu'un commit contient au final l'arborescence (objets git) du projet au moment de la création du commit en question.

Un commit contient toujours le pointeur sur l'objet qui correspond au dossier racine du projet.

EXEMPLE (GOT):

1. `git log`
2. noter le sha1 du commit
3. Aller dans `.git/objects/` et retrouver le sha1 du commit (2 premiers chars = dossier, reste = fichier)

Visualiser l'historique

```
commit 8a16f0301c192603761047146df513de21b985c2 (HEAD -> master, tag: MON_SITE_V5)
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 19:05:35 2018 +0200

    ajout médaille d'or

commit bd88f16cf89d6c2bbd6f0776567fb915683a70 (origin/master)
Merge: 92df0e3 6d73307
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:47:54 2018 +0200

    merge de ma branche BRANCH A MERGE avec ma master

commit 92df0e310f1bce1023e8aecdf7e78d7a919dbba9
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:04:28 2018 +0200

    ajout commande git pull

commit 6d73307cb5c9581d802816ea455c73d631eb3385 (origin/BRANCH_A_MERGE)
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:20:01 2018 +0200

    ajout git log

commit 5f0e4576815501bc3bce4707c5de4f45ea07a994 (BRANCH_A_MERGE)
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:06:28 2018 +0200

    ajout git push

commit d4d4451ec7cf1186e312c341d91a0a3d0fa44665
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:04:28 2018 +0200

    ajout commande git pull

commit e0a68a7debbe02994696bd0d04719b3e797cbe14 (tag: MON_SITE_V4)
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 16:16:56 2018 +0200

    ajout médaille argent
```

git log

```
commit 8a16f0301c192603761047146df513de21b985c2 (HEAD -> master, tag: MON_SITE_V5)
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 19:05:35 2018 +0200

    ajout médaille d'or

commit bd88f16cf89d6c2bbd6f0776567fb915683a70 (origin/master)
Merge: 92df0e3 6d73307
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:47:54 2018 +0200

    merge de ma branche BRANCH A MERGE avec ma master

commit 92df0e310f1bce1023e8aecdf7e78d7a919dbba9 (origin/BRANCH_A_MERGE)
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:20:01 2018 +0200

    ajout git log

commit 5f0e4576815501bc3bce4707c5de4f45ea07a994 (BRANCH_A_MERGE)
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:06:28 2018 +0200

    ajout git push

commit d4d4451ec7cf1186e312c341d91a0a3d0fa44665
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:04:28 2018 +0200

    ajout commande git pull

commit 92df0e310f1bce1023e8aecdf7e78d7a919dbba9
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:04:28 2018 +0200

    ajout commande git pull

commit e0a68a7debbe02994696bd0d04719b3e797cbe14 (tag: MON_SITE_V4)
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 16:16:56 2018 +0200

    ajout médaille argent
```

git log --graph

```
git log [ref/sha1]           // Par défaut HEAD
git log --graph [ref/sha1]  // Par défaut HEAD
```

Pour afficher l'historique d'un dépôt, il faut utiliser la commande "git log". Il est possible d'ajouter l'option "--graph" pour un affichage plus graphique.

Voir l'historique en résumé

```
8a16f0301c192603761047146df513de21b985c2 (HEAD -> master, tag: MON_SITE_V5) ajout médaille d'or
bd88f16cf89d6cbc2bbd6f0776567fb915683a70 (origin/master) merge de ma branche BRANCH_A MERGE avec ma master
92df0e310f1bce1023e8aecdf7e78d7a919dbba9 ajout commande git pull
6d73307cb5c9581d802816ea455c73d631eb3385 (origin/BRANCH_A_MERGE) ajout git log
5f0e4576815501bc3bce4707c5de4f45ea07a994 (BRANCH_A_MERGE) ajout git push
d4d4451ec7cf1186e312c341d91a0a3d0fa44665 ajout commande git pull
e0a68a7debb02994696bd0d04719b3e797cbe14 (tag: MON_SITE_V4) ajout médaille argent
8dbc1c39264af7ed302de2346256ae4488dd8d2f supprimer le filtre sur les fichiers xml
584bdf56b90efd13e934d519e9659729f0f6352a nettoyage du fichier
af6361b380074f135c6b5e2e0baed1fe96885a0b merge de mes commits
8d4aac4434bbc5ad0e182a71db1bce0b7bb62a25 exclusion des fichiers xml
31acf90cfd13d759e73002bffdccf0f0bf359f1 exclusion des fichiers txt
35b795d2da9f8ad0b708f4fe4b65d69ed319a4c9 filtrage sur le dossier tmp
255dcffcb041af6ad0c92454bc034dd469b16580 ajout fichier gitignore
839cb3e1930386badc28d70890e1612c215cfd91 (tag: MON_SITE_V3) #2: ajout medaille bronze
71d33d22b5999db65c6db820bce785d5c28a2b2d #1: ajout du readme
2f30431505f4148a04be67b2af3730a689c3a2a0 ajout d'un gist
fbo0a495ffdd395c80f61ef9e90be064a682d03e8 (tag: MON_SITE_V2) ajout des titres
05196405e77e13936288bc0fc712f752a922d77b ajout description
f295ec73786821a5973450ebeb7449e2ed4d7613 (tag: MON_SITE_V1) ajout du top banner
fbcbf81201bdeb4547b7bec3c02ebf6d3f019
```

git log --oneline

git log --oneline [ref/sha1] // Une ligne par commit

l'option "--oneline" permet d'avoir un résumé des commits (sha1 + message + tags et pointeurs) en une seule ligne par commit

EXEMPLE (GOT):

- git log --graph --oneline
- git log --graph --oneline master
- git log --graph --oneline HEAD

Voir l'historique d'un fichier

```
commit 8a16f0301c192603761047146df513de21d985c2 (HEAD -> master, tag: MON_SITE_V5)
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 19:05:35 2018 +0200

    ajout médaille d'or

commit bd88f16cf89d6cbc2bbd6f0776567fb915683a70 (origin/master)
Merge: 92df0e3 6d73307
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:47:54 2018 +0200

    merge de ma branche BRANCH_A MERGE avec ma master

commit 92df0e310f1bce1023e8aedffe78d7a919dbba9
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:04:28 2018 +0200

    ajout commande git pull

commit 6d73307cb5c9581d802816ea455c73d631eb3385 (origin/BRANCH_A_MERGE)
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:20:01 2018 +0200

    ajout git log

commit 5f0e4576815501bc3bce4707c5de4f45ea07a994 (BRANCH_A_MERGE)
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:06:28 2018 +0200

    ajout git push

commit d4d4451ec7cfl186e312c341d91a0a3d0fa44665
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 17:04:28 2018 +0200

    ajout commande git pull

commit e0a68a7debbe02994696bdd04719b3e797cbe14 (tag: MON_SITE_V4)
Author: clone <arnaud.mercier.formation@gmail.com>
Date: Wed Jun 13 16:16:56 2018 +0200

    ajout médaille
```

git log hello.html

git log [ref/sha1] [file_name] // Depuis HEAD par défaut

Il est possible de filtrer le log pour afficher uniquement les commits qui modifient un fichier donné. Pratique pour voir l'historique d'un fichier en particulier

EXEMPLE (GOT):

- **git log --graph --oneline .gitignore**
- **git log --graph --oneline .gitignore HEAD**
- **git log --graph --oneline .gitignore master**

Voir les détails d'un commit

```
$ git show 8a16f0301c192603761047146df513de21b985c2
commit 8a16f0301c192603761047146df513de21b985c2 (HEAD -> master, tag: MON_SITE_V5)
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date:   Wed Jun 13 19:05:35 2018 +0200

    ajout médaille d'or

diff --git a/hello.html b/hello.html
index 0a4d4c7..816729a 100644
--- a/hello.html
+++ b/hello.html
@@ -26,7 +26,7 @@
     <!-- recompense -->
     <div>
         <h1> Mes récompenses </h1>
-        <p>  
+        <p>  
     </div>
     <!-- mon gist -->
```

git show 8a16f03

git show [sha1/ref] // Par défaut HEAD

Pour afficher les détails d'un commit en particulier, il est possible d'utiliser la commande "git show" en lui indiquant le sha1 du commit en question

EXEMPLE (GOT):

- **git show master**

Voir l'historique en détail

```
$ git show 8a16f0301c192603761047146df513de21b985c2
commit 8a16f0301c192603761047146df513de21b985c2 (HEAD -> master, tag: MON_SITE_V5)
Author: Arnaud Mercier <arnaud.mercier.formation@gmail.com>
Date:   Wed Jun 13 19:05:35 2018 +0200

    ajout médaille d'or

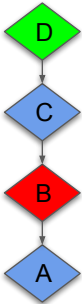
diff --git a/hello.html b/hello.html
index 0a4d4c7..816729a 100644
--- a/hello.html
+++ b/hello.html
@@ -26,7 +26,7 @@
     <!-- recompense -->
     <div>
         <h1> Mes récompenses </h1>
-        <p>  
+        <p>  
     </div>
 <!-- mon gist -->
```

git log -p -1

git log -p -[N] // -N pour limiter aux N derniers commits

l'option -p pour avoir les détails et l'option -N (n est un nombre) pour afficher uniquement les n derniers commits

Voir les évolutions du code



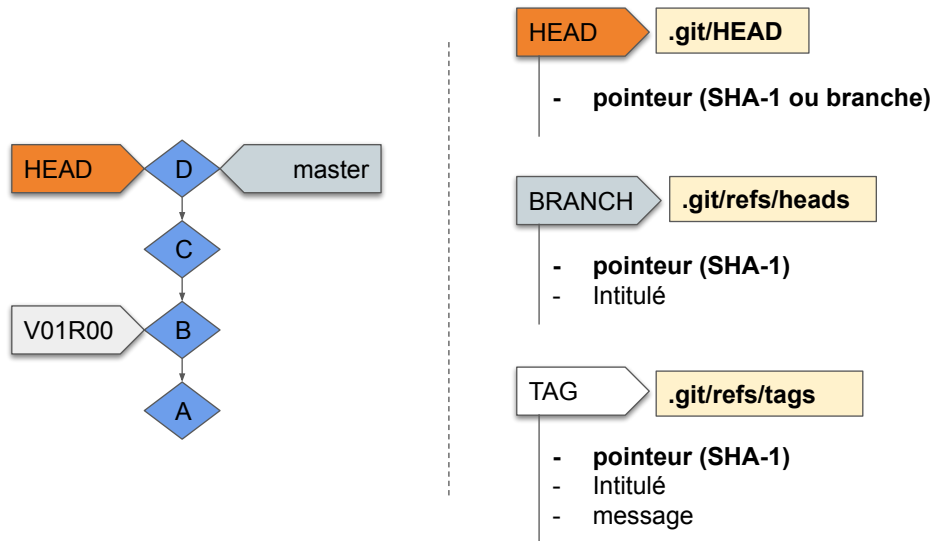
```
$ git diff 92df0e310f1bce1023e8aecdf7e78d7a919dbba9 af6361b380074f135c6b5e2e0baed1fe96885a0b
diff --git a/.gitignore b/.gitignore
index cdf8a87..ae7e9eb 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1,2 +1,4 @@
+tmp/todo.txt
+tmp/*
+*.txt
diff --git a/hello.html b/hello.html
index e733468..ed7a15d 100644
--- a/hello.html
+++ b/hello.html
@@ -26,7 +26,7 @@
<!-- recompense -->
<div>
  <h1> Mes récompenses </h1>
  <p>   </p>
+  <p>  </p>
</div>
<!-- mon gist -->
<p> <script src="https://gist.github.com/amgitformation/6ffb2f05ea6ad2a58ef129a15541c978.js">
</div>
<!-- commandes git -->
<div>
  <h1> Mes commandes git </h1>
  <p> git pull </p>
</div>
</body>
</html>
```

diff entre deux commits

```
git diff [sha1 / ref] // Comparaison HEAD et la référence
git diff [sha1 / ref] [sha1 / ref] // Comparaison entre 2 références
```

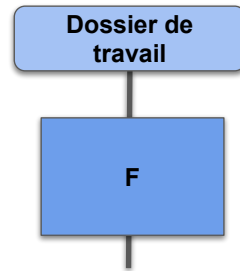
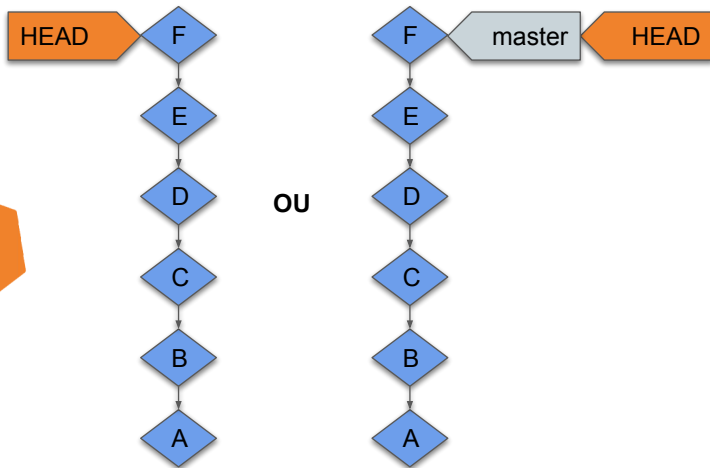
La commande “git diff” peut également être utilisée pour comparer des versions d’un fichier dans l’historique.

Les références locales



Nous allons voir les références locales. Elles correspondent à des pointeurs sur des commits, ou dit autrement, à un point d'entrée sur l'arborescence git

Pointeur HEAD



HEAD = REF

Le pointeur HEAD indique le commit sur lequel nous nous trouvons actuellement, celui qui est utilisé comme référence pour notre espace de travail

Pour se déplacer dans l'historique, il faut alors déplacer ce pointeur.

HEAD peut soit pointer sur un commit ou sur une branche. Dans ce second cas, la branche en question est dite active.

Se déplacer dans l'historique



Se déplacer dans l'historique d'un dépôt git correspond au fait de déplacer le pointeur HEAD sur le commit souhaité. Pour cela, on utilise la commande "git checkout sha1" ou "git checkout pointeur": exemple git checkout master

EXEMPLE (GOT):

1. `git log --oneline master`
2. `git checkout [commit]`
3. `git log --oneline master`
4. `git checkout master`
5. `git log --oneline master`

Références relatives et absolues



```
git checkout [ref/sha1]^ // On remonte un parents par ^ depuis la ref  
git checkout [ref/sha1]~n // On remonte N parents depuis la ref
```

Il est aussi possible de donner une référence en relative:

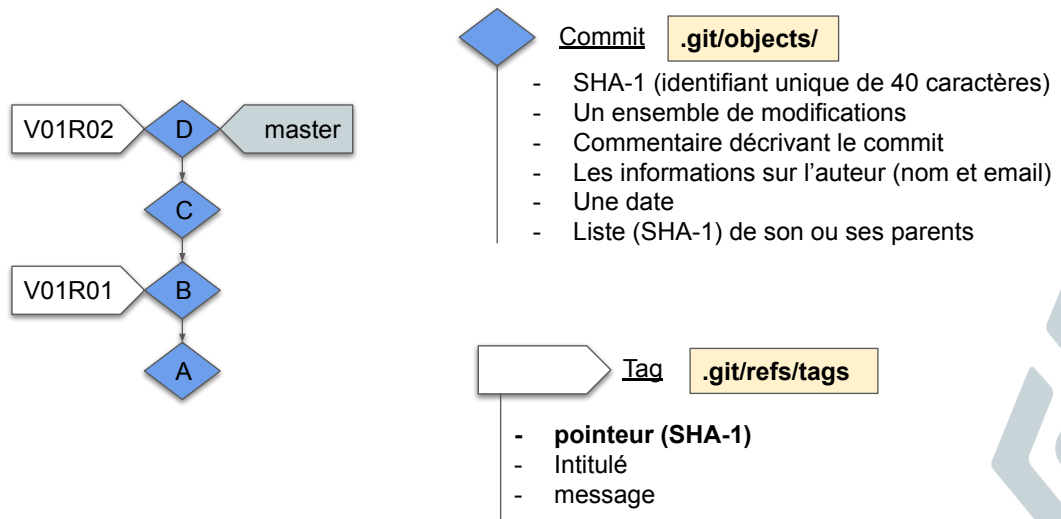
^: on remonte d'un parent par ^

~N: on remonte de N parent

EXEMPLE (GOT):

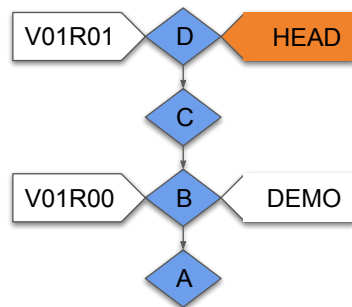
1. `git log --oneline master`
2. `git checkout master^`
3. `git log --oneline master`
4. `git checkout master~2`
5. `git log --oneline master`
6. `git checkout master`
7. `git log --oneline master`

Les tags personnalisés



Les tags sont également des pointeurs (références). Ils pointent également sur un commit mais n'en bougent pas. Ils permettent en fait de marquer un commit comme étant une version remarquable de votre historique. Par exemple une release client. Ces tags comportent alors le sha1 du commit pointé ainsi que d'autres informations comme par exemple un intitulé

Créer des tags



```
git checkout D  
git tag V01R01
```

```
git tag V01R00 [sha1 B]
```

```
git tag DEMO [sha1 B]
```

```
git tag [MON_TAG] [sha1 / ref] // Ajout d'un tag sur le commit référencé (message du commit)  
git tag [MON_TAG] [sha1 / ref] -m"message" // Ajout d'un message personnalisé sur le tag
```

Vous pouvez créer des tags sur un commit ou utiliser une référence pour obtenir le sha1 du commit.

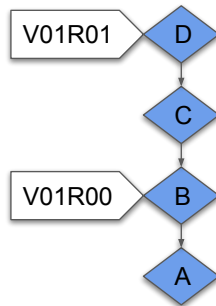
Il est possible d'avoir plusieurs tags sur un commit donné

Sans paramètre de référence, le tag est mis sur le commit courant (HEAD par défaut)

EXEMPLE (GOT):

1. **git checkout master**
2. **git tag SAISON_2 -m"Meilleur saison"**
3. **git tag SAISON_1 [SHA1 commit plus ancien]**
4. **git log --oneline**

Lister les tags



```
git tag  
V01R00  
V01R01
```

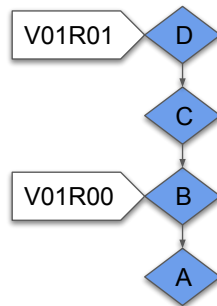
```
git tag           // Affiche la liste des tags (nom)  
git tag -n        // Affiche la liste des tags (nom + message)  
git tag -l "V01R*" // Affiche la liste des tags qui correspondent au filtre
```

Pour visualiser la liste des tags présente dans votre dépôt, vous pouvez utiliser la commande “git tag” l’options -l permet de filtrer les résultats via des regexs

EXEMPLE (GOT):

1. `git tag -n`
2. montrer le contenu de `.git/refs/tags`

Supprimer des tags



`git tag -d V01R01`

`git tag -d [TAG]`

Pour supprimer un tag, il faut toujours utiliser la commande `git tag` mais avec l'option `-g` suivi du nom du tag

EXEMPLE (GOT):

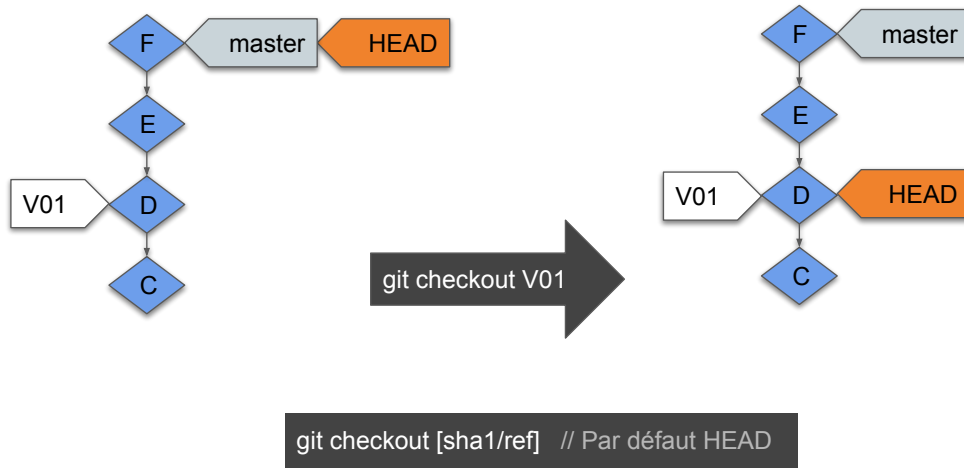
1. `git tag -n`
2. `git tag TEST SAISON_1`
3. `git log`
4. `git tag -d TEST`
5. `git log`

Déplacer des tags



Il est possible de déplacer un tag via l'option -f. Cela permet aussi de modifier le message associé

Se déplacer dans l'historique via un Tag

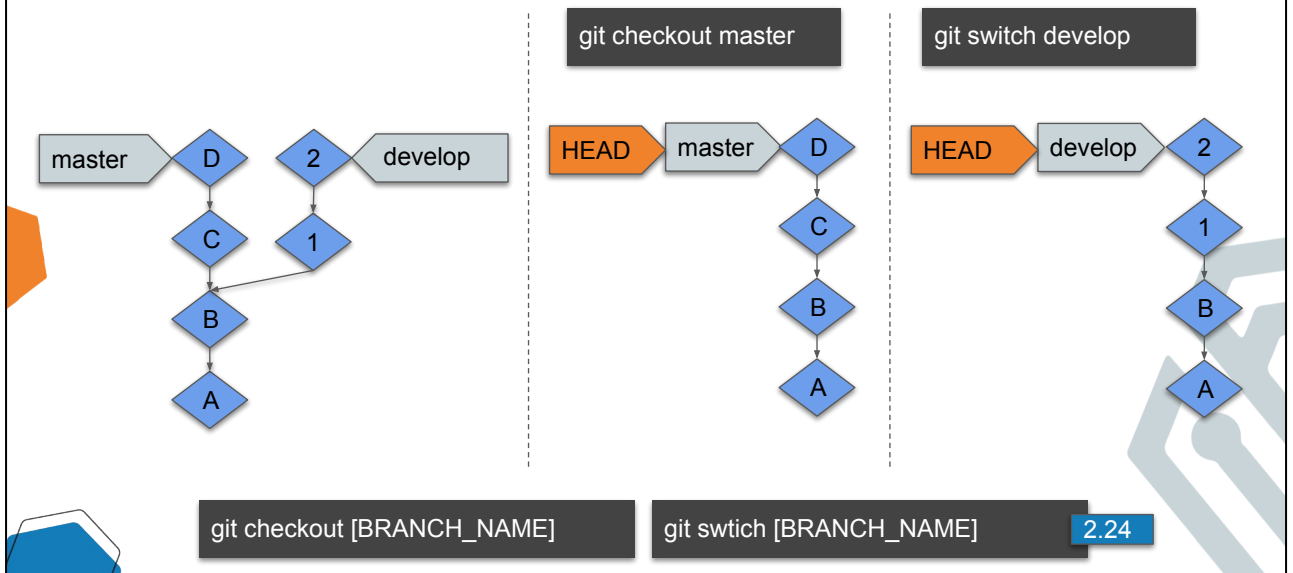


Nous pouvons utiliser le tag comme une référence pour nos déplacements dans l'historique

EXEMPLE (GOT):

1. `git checkout SAISON_1`
2. `git log`
3. `git log master`
4. `git checkout master`

Le système de branches



Une branche est simplement un pointeur, du nom de la branche, sur un commit. Une branche est alors constitué de ce commit pointé et de ses parents.


Il est recommandé d'utiliser `git switch` pour activer une branche plutôt que `checkout`

EXEMPLE (GOT):

1. `git checkout SAISON_1`
2. `git switch master`

Branche courante



 Branche principale 'master' devient 'main'

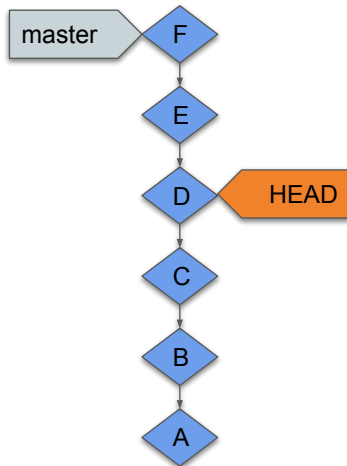
Un ensemble de commit liés entre eux constituent une arborescence Git. Une branche est un simple pointeur sur un commit. On dit que cette référence est la tête d'un historique. En effet un historique est portée par une branche qui pointe sur le haut de l'historique. Nous avons ensuite dans cet historique l'ensemble des commits associés par parenté.

La branche par défaut d'un dépôt git est la branche master ou main

Lorsque HEAD pointe sur une branche, cette dernière est alors active. Cela implique que tout changement de l'arborescence (comme par exemple un nouveau commit), sera inclus dans cette branche.

Dans le cas d'un nouveau commit dans une branche, le pointeur de branche se déplacera sur ce nouveau commit

Detached HEAD



```
$ git checkout 8a16f0301c192603761047146df513de21b985c2
Note: checking out '8a16f0301c192603761047146df513de21b985c2'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 8a16f03 ajout médaille d'or
```

```
$ git status
HEAD detached at 8a16f03
nothing to commit, working tree clean
```

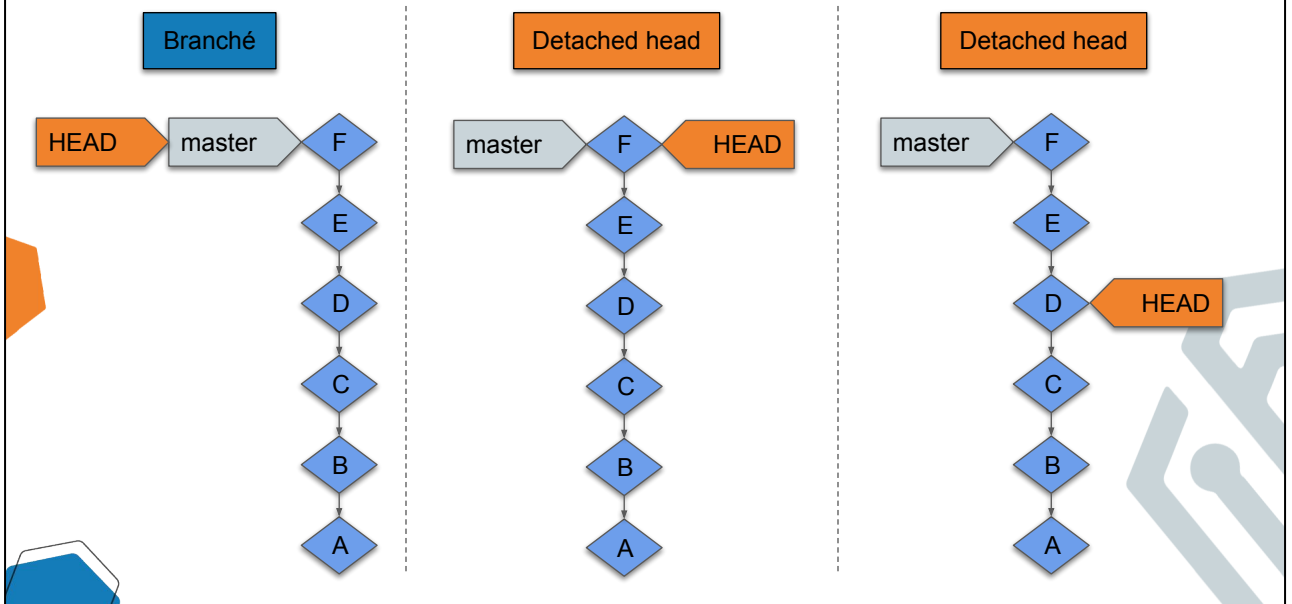
```
git checkout [BRANCH_NAME]
```

```
git switch [BRANCH_NAME]
```

2.24

L'état detached head correspond au fait de se retrouver sur un commit qui n'est pas pointé directement par un pointeur de branche. Pour remédier à cela, il faut se positionner sur une référence de branche: ex `git checkout master`

Detached HEAD

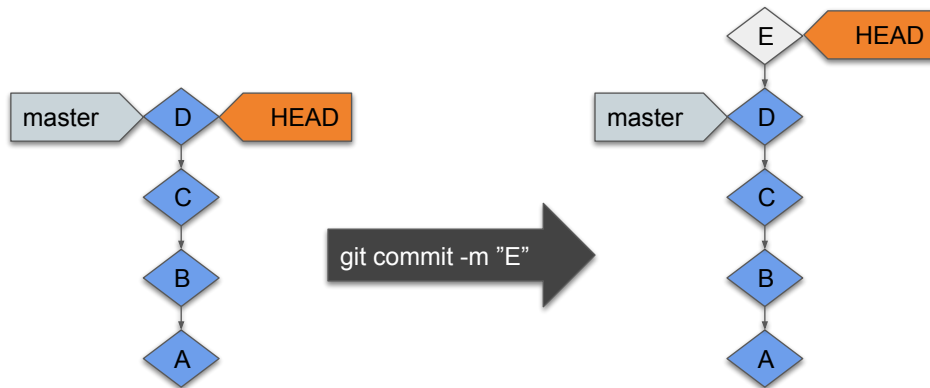


Si aucune branche n'est active, on est en detached head. HEAD pointe sur un commit et non une branche. Même si HEAD est sur le même commit que la branche, cela n'active pas la branche

EXEMPLE (GOT):

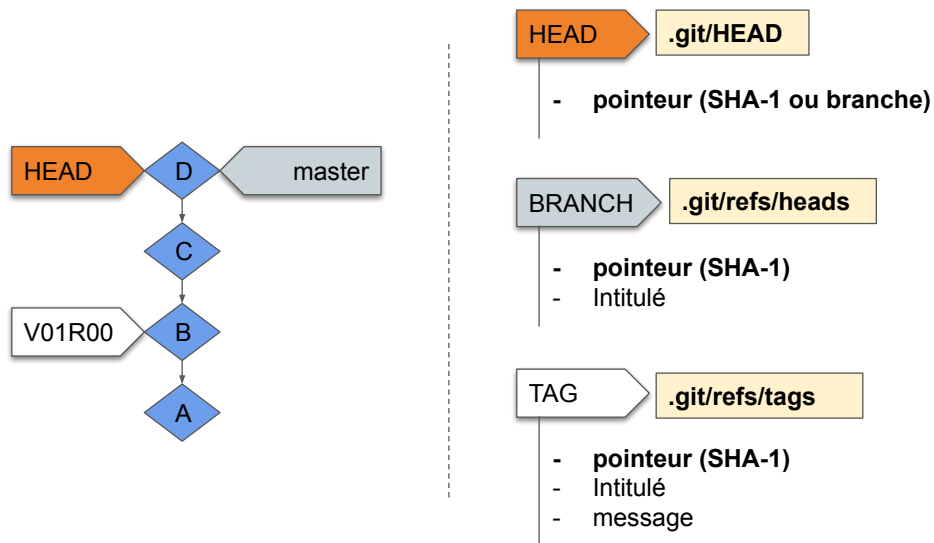
1. `git checkout master^`
2. `git log --oneline master`
3. `git status`
4. `git switch master`
5. `git status`

Commits orphelins



Attention, il est possible de faire des commits dans ce cas mais ils seront orphelin (non présent dans une branche)

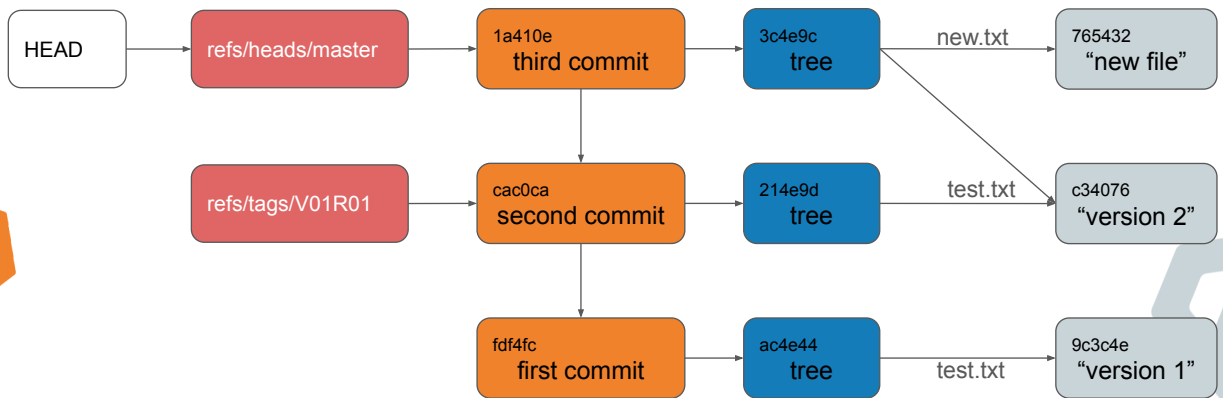
Les références locales



Nous avons vu les 3 références locales:

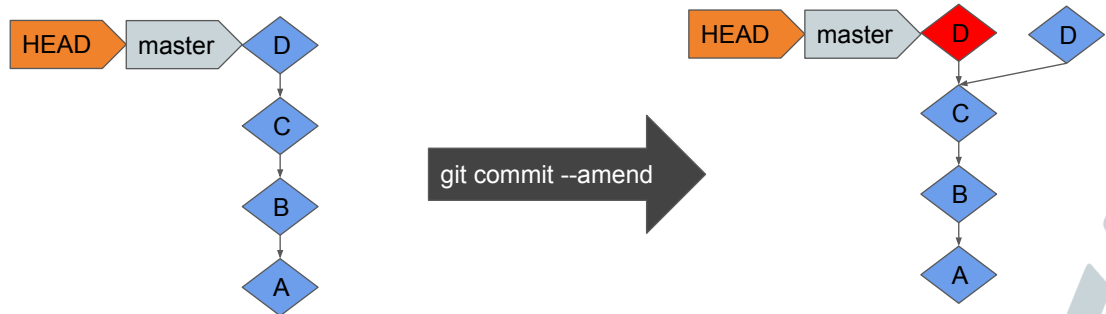
- **HEAD:** Correspond à notre référence pour l'espace de travail: commit ou branche
- **BRANCH:** Correspond à la tête d'un historique (contexte de travail)
- **TAG:** Correspond à un alias qui permet d'identifier humainement un commit

Les références locales



Voici une vue détaillée des relations entre les références et les commits

Rien ne se perd !



Git ne supprime et modifie pas les commits.

Lors d'un amend par exemple, git vas créer un nouveau commit et faire pointer la branche sur ce dernier

EXEMPLE (GOT):

1. `git log --oneline`
2. `git commit --amend`
3. `git log --oneline --graph master sha1_orphelin`

Créer l'inverse d'un commit



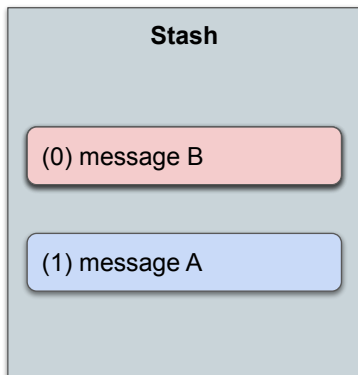
Vous avez fait un commit que vous souhaitez faire les modifications pour annuler ce dernier? et bien la commande "git revert" est la solution. elle permet d'appliquer l'inverse du commit passé en paramètre.

Attention: cela ne supprime pas le commit de l'historique mais permet de créer un autre commit qui est l'inverse de celui passé en paramètre

EXEMPLE (GOT):

1. `git log --oneline`
2. `git revert [sha1_ajout_limier]`
3. `git log --oneline`

La zone de stash



```
git stash save "message" // Empile une entrée au stash
git stash list           // Liste les entrées du stash
git stash pop [N]       // Dépile l'entrée N du stash
git stash pop           // Dépile la 1ere entrée du stash
git stash clear         // Vider le stash
```

la zone de stash est une zone à part qui vous permet de mettre de côté des modifications dans une pile pour les conserver. Il est alors possible de les réappliquer dans votre espace de travail par la suite.

Pour ajouter vos modifications courantes dans la zone de stash, il faut utiliser la commande "git stash save"

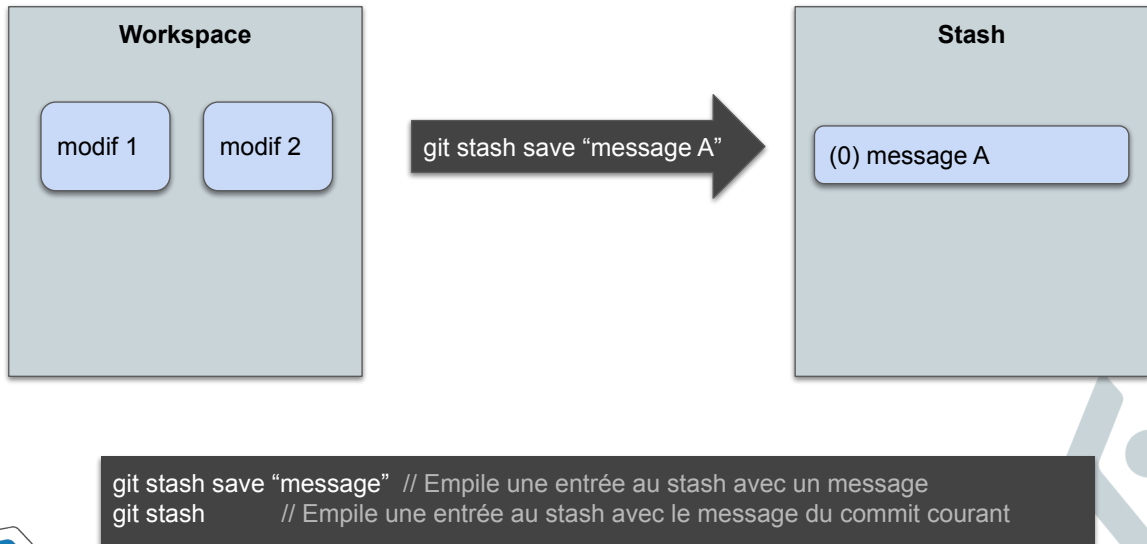
Pour lister les stashes: git stash list

pour récupérer une stash: git stash pop N (n est la place de la stash dans la pile)

comme la zone de stash est une pile, il est possible de simplement utiliser les commande "git stash" pour ajouter un stash au dessus de la pile et la commande "git stash pop" pour récupérer la stash du dessus de la pile.

cette commande est très utile comme par exemple dans le cas ou vous travaillez sur une correction de bug et que vous souhaitez revenir a une version plus ancienne de votre code pour voir si il était déjà présent. Vous pouvez alors mettre de côté vos modifications, faire le "git checkout" puis revenir et réappliquer le stash

La zone de stash



la zone de stash est une zone à part qui vous permet de mettre de côté des modifications dans une pile pour les conserver. Il est alors possible de les réappliquer dans votre espace de travail par la suite.

Pour ajouter vos modifications courantes dans la zone de stash, il faut utiliser la commande "git stash save"

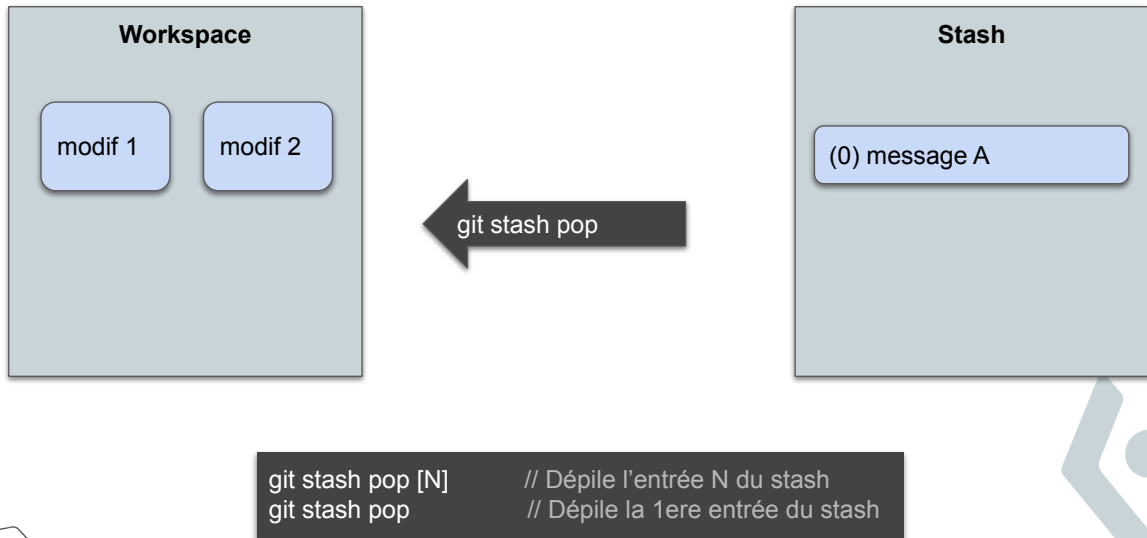
Pour lister les stashes: git stash list

pour récupérer une stash: git stash pop N (n est la place de la stash dans la pile)

comme la zone de stash est une pile, il est possible de simplement utiliser les commande "git stash" pour ajouter un stash au dessus de la pile et la commande "git stash pop" pour récupérer la stash du dessus de la pile.

cette commande est très utile comme par exemple dans le cas ou vous travaillez sur une correction de bug et que vous souhaitez revenir a une version plus ancienne de votre code pour voir si il était déjà présent. Vous pouvez alors mettre de côté vos modifications, faire le "git checkout" puis revenir et réappliquer le stash

La zone de stash



la zone de stash est une zone à part qui vous permet de mettre de côté des modifications dans une pile pour les conserver. Il est alors possible de les réappliquer dans votre espace de travail par la suite.

Pour ajouter vos modifications courantes dans la zone de stash, il faut utiliser la commande “git stash save”

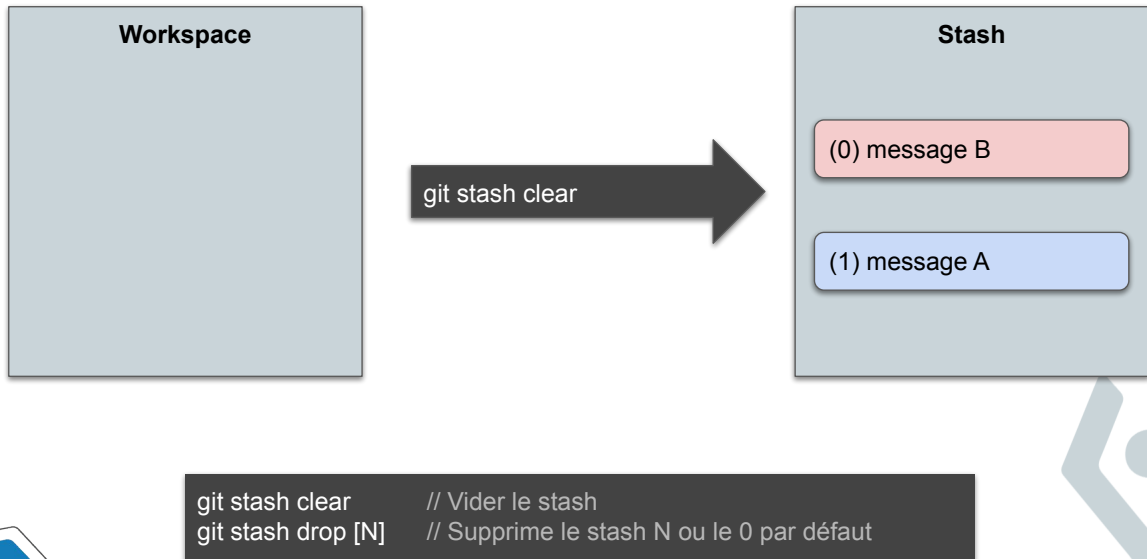
Pour lister les stashes: git stash list

pour récupérer une stash: git stash pop N (n est la place de la stash dans la pile)

comme la zone de stash est une pile, il est possible de simplement utiliser les commande “git stash” pour ajouter un stash au dessus de la pile et la commande “git stash pop” pour récupérer la stash du dessus de la pile.

cette commande est très utile comme par exemple dans le cas ou vous travaillez sur une correction de bug et que vous souhaitez revenir a une version plus ancienne de votre code pour voir si il était déjà présent. Vous pouvez alors mettre de côté vos modifications, faire le “git checkout” puis revenir et réappliquer le stash

La zone de stash



la zone de stash est une zone à part qui vous permet de mettre de côté des modifications dans une pile pour les conserver. Il est alors possible de les réappliquer dans votre espace de travail par la suite.

Pour ajouter vos modifications courantes dans la zone de stash, il faut utiliser la commande "git stash save"

Pour lister les stashes: git stash list

pour récupérer une stash: git stash pop N (n est la place de la stash dans la pile)

comme la zone de stash est une pile, il est possible de simplement utiliser les commande "git stash" pour ajouter un stash au dessus de la pile et la commande "git stash pop" pour récupérer la stash du dessus de la pile.

cette commande est très utile comme par exemple dans le cas ou vous travaillez sur une correction de bug et que vous souhaitez revenir a une version plus ancienne de votre code pour voir si il était déjà présent. Vous pouvez alors mettre de côté vos modifications, faire le "git checkout" puis revenir et réappliquer le stash

TP 4.1

Scénario

Vous avez oublié de poser le tag SITE_V1 sur le commit qui modifie le message de bienvenue sur le site.

Le problème c'est que vous avez des modifications en cours dans le WS.

Objectifs

- 1- Posez le tag SITE_V01 avec le message "Livraison de la Beta au client"
- 2- Vous déplacer sur SITE_V01 pour tester cette version
- 3- Revenir à l'état dans lequel vous étiez avant ces manipulations.
- 4- Faire le commit avec le message "ajout de la partie recompenses"
- 5- Posez le tag SITE_V02

CORRECTION:

1. Comme nous avons des modifications dans l'espace de travail, il n'est pas possible de checkout:
 - a. `git tag SITE_V01 6d89e9c -m"Livraison de la Beta au client"`
2. Il faut dans un premier temps mettre de côté les modifications:
 - a. `git stash`
 - b. `git checkout SITE_V01`
3. Il faut réactiver la branche et réappliquer le stash:
 - a. `git switch master`
 - b. `git stash pop`
4. `git commit -m"ajout de la partie recompenses"`
5. `git tag SITE_V02`

TP 4.2

Scénario

Après avoir posé votre tag sur le commit qui correspond à la livraison 2 du client, vous vous apercevez que vous avez oublié de commit une modification.

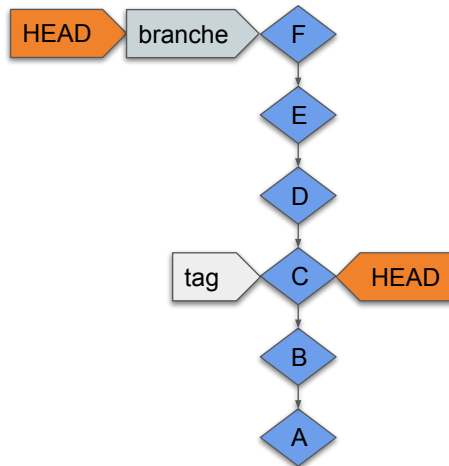
Objectifs

- 1- Faire un nouveau commit "ajout de la médaille de bronze"
- 2- Déplacer le tag SITE_V02 sur ce dernier en ajoutant comme message "Maquette pour le salon".
- 3- Comparer les deux versions taggées

CORRECTION:

1. `git commit -am"ajout de la médaille de bronze"`
2. On peut soit forcer le tag avec l'option "-f", soit:
 - a. `git tag -d SITE_V02`
 - b. `git tag SITE_V02`
3. `git diff SITE_V01 SITE_V02`

Bilan



```
git log [--oneline --graph]
```

```
git checkout [sha1 / ref]
```

```
git diff [sha1 / ref]
```

```
git show [sha1 / ref]
```

```
git tag [-d] [nom tag]
```

```
git stash [pop]
```

Nous avons vu comment visualiser l'historique et comment naviguer dans ce dernier via la commande checkout

Nous avons également vu les 3 références locales:

- HEAD: Correspond à notre référence pour l'espace de travail: commit ou branche
- BRANCH: Correspond à la tête d'un historique (contexte de travail)
- TAG: Correspond à un alias qui permet d'identifier humainement un commit