

Naviguer dans l'historique

Table des matières

I. Contexte	3
II. Usages basiques de git log	3
III. Exercice : Appliquez la notion	7
IV. Formater l'affichage de git log	7
V. Rechercher, comprendre et filtrer l'historique	9
VI. Exercice : Appliquez la notion	11
VII. Auto-évaluation	11
A. Exercice final	11
B. Exercice : Défi	12
Solutions des exercices	13

I. Contexte

Durée : 1 h

Environnement de travail : GitBash / Terminal

Pré-requis : Bases de Git

Contexte

Utiliser un outil de version comme Git au cours des développements d'un projet est considéré comme une bonne pratique, car il permet de conserver une trace des modifications réalisées sur le projet et d'alterner entre différentes versions et états. Cela implique que, plus un projet avance, plus les modifications réalisées sont nombreuses, et donc plus il devient difficile d'en lister les impacts.

En effet, avec le temps, l'historique devient de plus en plus important et compliqué à interpréter : nous allons voir ici les outils que Git met à notre disposition pour visualiser les modifications réalisées au cours du temps en décrivant en détails l'utilisation de la commande `git log`. Nous aborderons tout d'abord son utilisation basique avant d'aborder les possibilités plus avancées que cette commande propose, pour finalement présenter une solution permettant d'en faciliter les usages.

II. Usages basiques de git log

Objectif

- Apprendre à afficher l'historique de Git

Mise en situation

Il est courant, lors des développements, d'avoir besoin de connaître les modifications apportées à un projet versionné avec Git.

Pour cela, cet outil dispose de la commande `git log`, permettant d'afficher la liste des commits présents sur la branche en cours.

Nous allons voir quelques-unes de ses utilisations en présentant l'utilisation de base, ainsi que les options permettant de limiter le nombre de commits, ou les modifications qu'ils contiennent.

Méthode Afficher l'historique des modifications

La commande `git log` permet d'afficher les commits sur la branche courante en les ordonnant du plus récent au plus ancien.

L'affichage des résultats présente, pour chaque commit, sa somme de contrôle SHA-1, ainsi que la branche sur lequel il est positionné, son auteur, sa date et son commentaire.

Exemple

```
1 git log
```

```
$ git log
commit 547164acf20b297229b8592a99043b72016e9c24 (HEAD -> master)
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:24:15 2020 +0200

    Nouvel affichage

commit 50d170c325b1d798d2029f17a4e3d28fbd5d5653
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:23:22 2020 +0200

    Modification du texte

commit 7a9e9b26551335e5663f7975d4c9ea090e167ca6
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 11:04:40 2020 +0200

    Ajout du fichier index

commit 56eb436b9f47f772226801a30efc3b70f5825212
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:05:09 2020 +0200

    Affichage du texte

commit 9a8f13ca400f10057cd1a2802c04f22417d29276
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:02:24 2020 +0200

    Ajout du fichier main
```

L'utilisation de la commande `git log` affiche tous les commits de la branche, du plus récent au plus ancien.

Méthode

Afficher l'historique et le détail de toutes les modifications

Pour certains cas d'application, consulter la liste des commits peut ne pas suffire à couvrir l'ensemble des besoins. Il peut par exemple être nécessaires de visualiser directement le détail des modifications apportées par les commits précédents.

Pour obtenir cette information, la commande `git log` dispose de l'option `-p`.

Cette option permet de cumuler l'utilisation de la commande `git log` et de la commande `git diff`, appliquées à chaque commit obtenu.

Exemple

```
1 git log -p
```

```
$ git log -p
commit 547164acf20b297229b8592a99043b72016e9c24 (HEAD -> master)
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:24:15 2020 +0200

    Nouvel affichage

diff --git a/main.php b/main.php
index bd4caff..9183e5a 100644
--- a/main.php
+++ b/main.php
@@ -1,2 +1,3 @@
 <?php
-var_dump('Hello world');
\ No newline at end of file
+var_dump('Hello world');
+var_dump('Je suis un test');
\ No newline at end of file

commit 50d170c325b1d798d2029f17a4e3d28fbd5d5653
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:23:22 2020 +0200

    Modification du texte

diff --git a/main.php b/main.php
index 9749d65..bd4caff 100644
--- a/main.php
+++ b/main.php
@@ -1,2 +1,2 @@
 <?php
-var_dump('Hello world');
\ No newline at end of file
+var_dump('Hello world');
\ No newline at end of file
```

L'utilisation de la commande `git log -p` présente pour chaque commit les éléments modifiés en préfixant par `-` les suppressions, et par `+` les ajouts.

Méthode Afficher l'historique et un résumé des modifications

Si l'affichage détaillé des modifications réalisées par chaque commit est trop conséquent pour être lisible, `git log` offre la possibilité d'en résumer le contenu grâce à la commande `--stat`.

Cette option permet de visualiser rapidement le nombre de modifications intervenues sur chaque fichier concerné par chaque commit.

Exemple

```
1 git log --stat
```

```
$ git log --stat
commit 547164acf20b297229b8592a99043b72016e9c24 (HEAD -> master)
Author: JohnDoeTest <JohnDoeTest@test.com>
Date:   Fri Apr 3 10:24:15 2020 +0200

    Nouvel affichage

    main.php | 3 +-
    1 file changed, 2 insertions(+), 1 deletion(-)

commit 50d170c325b1d798d2029f17a4e3d28fbd5d5653
Author: JohnDoeTest <JohnDoeTest@test.com>
Date:   Fri Apr 3 10:23:22 2020 +0200

    Modification du texte

    main.php | 2 +-
    1 file changed, 1 insertion(+), 1 deletion(-)

commit 7a9e0b26351335e5663f7975d4c9ea090e167ca6
Author: JohnDoeTest <JohnDoeTest@test.com>
Date:   Fri Apr 3 11:04:40 2020 +0200

    Ajout du fichier index

    index.php | 0
    1 file changed, 0 insertions(+), 0 deletions(-)

commit 56eb430b9f47f77226001a10efc3b70f5825212
Author: JohnDoeTest <JohnDoeTest@test.com>
Date:   Fri Apr 3 10:05:09 2020 +0200

    Affichage du texte

    main.php | 2 +-
    1 file changed, 2 insertions(+)
```

L'utilisation de `git log --stat` permet de visualiser de manière concise les modifications réalisées dans chaque commit.

Méthode Limiter le nombre de commits affichés

Avec le temps, le nombre de commits présents sur une branche peut devenir trop important pour rendre celui-ci lisible.

Il est possible de spécifier à la commande `git log` un nombre de commits auxquels remonter avec l'option `-n`, où `n` est le nombre de commits souhaités.

Exemple

```
1 git log -2
```

```
$ git log -2
commit 547164acf20b297229b8592a99043b72016e9c24 (HEAD -> master)
Author: JohnDoeTest <JohnDoeTest@test.com>
Date:   Fri Apr 3 10:24:15 2020 +0200

    Nouvel affichage

commit 50d170c325b1d798d2029f17a4e3d28fbd5d5653
Author: JohnDoeTest <JohnDoeTest@test.com>
Date:   Fri Apr 3 10:23:22 2020 +0200

    Modification du texte
```

L'utilisation de `git log -2` affiche les deux derniers commits présents sur la branche.

Remarque

Il est possible de cumuler les différentes options à la commande `git log` pour une meilleure gestion de son affichage.

Exemple

```
1 git log -p -2
```

```
commit 547164acf20b297229b8592a99043b72016e9c24 (HEAD -> master)
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:24:15 2020 +0200

    Nouvel affichage

diff --git a/main.php b/main.php
index bd4caff..9183e5a 100644
--- a/main.php
+++ b/main.php
@@ -1,2 +1,3 @@
<?php
-var_dump('Hello world');
\ No newline at end of file
+var_dump('Hello world');
+var_dump('Je suis un test');
\ No newline at end of file

commit 50d17bc325b1d798d7029f17a4e3d28fbd5d5653
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:23:22 2020 +0200

    Modification du texte

diff --git a/main.php b/main.php
index 9749df5..bd4caff 100644
--- a/main.php
+++ b/main.php
@@ -1,2 +1,2 @@
<?php
-var_dump('Hello world');
\ No newline at end of file
+var_dump('Hello world');
\ No newline at end of file
```

En cumulant l'option `-p` et `-{n}` à la commande `git log`, l'historique des modifications apportées par les deux derniers commits est affiché.

Méthode Définir des alias

Il peut être fastidieux, lorsque l'on travaille avec Git, de devoir se rappeler ou de saisir entièrement des commandes régulièrement utilisées.

Heureusement, Git offre la possibilité de créer des alias, ce qui permet de ne devoir taper qu'une commande plus courte pour réaliser l'action souhaitée.

Pour cela, il faut utiliser la commande `git config --global alias.{nom_alias} {commande_git}`.

Exemple

```
1 git config --global alias.logmodif 'log -p'
2 git logmodif
```

```
$ git logmodif
commit 547164acf20b297229b8592a99043b72016e9c24 (HEAD -> master)
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:24:15 2020 +0200

    Nouvel affichage

diff --git a/main.php b/main.php
index bd4caff..9183e5a 100644
--- a/main.php
+++ b/main.php
@@ -1,2 +1,3 @@
<?php
-var_dump('Hello world');
\ No newline at end of file
+var_dump('Hello world');
+var_dump('Je suis un test');
\ No newline at end of file
```

La commande `git config --global alias.logmodif 'git log -p'` crée un alias global dans la configuration de Git nommé `logmodif` et renvoyant à la commande `git log -p`.

L'utilisation de `git logmodif` affiche bien les détails des modifications effectuées au sein de chaque commit.

Fondamental À retenir

- La commande `git log` permet de visualiser la liste des commits présents sur la branche en cours, du plus récent au plus ancien.
- Il est possible de lui passer l'option `-p` pour afficher les modifications apportées par chaque commit, ainsi que l'option `--stat` afin d'en obtenir un affichage plus concis.
- L'option `-{n}`, où `n` précise le nombre de commits à afficher, quant à elle limite l'affichage de l'historique au nombre de commits souhaités.

III. Exercice : Appliquez la notion

Question

[solution n°1 p.15]

Définissez un alias qui permettra d'afficher l'historique de votre projet, avec un résumé condensé des dernières modifications pour les 5 derniers commits.

IV. Formater l'affichage de git log

Objectif

- Perfectionner l'affichage de `git log`

Mise en situation

La commande `git log` est un outil majeur dans l'utilisation de Git, car elle permet de consulter l'historique des modifications conservées par celui-ci. Cela permet à chaque membre de l'équipe de consulter le contenu des différents commits rapidement.

En revanche, lorsque de trop nombreux commits ont été réalisés, il devient difficile d'en consulter l'historique.

Nous allons voir ici comment cette commande permet de transformer les résultats obtenus pour afficher un rendu plus accessible.

Méthode Améliorer la lisibilité de l'historique

Lorsque le journal d'historique affiché par la commande `git log` devient moins lisible, il est possible d'influer sur son affichage en utilisant l'option `--pretty`.

Cette option accepte différents paramètres de formatage préconfigurés, tels que `oneline` qui affiche les résultats sur une seule ligne, ou `short` ou `fuller` qui vont respectivement afficher de plus en plus d'informations.

Exemple

```
1 git log --pretty=oneline
```

```
$ git log --pretty=oneline
commit 58d17bc325b5d798d2029f17a4e3d28fbd5d5653 (HEAD -> master) Nouvel affichage
commit 7a9e9b26351335e5663f7975d4c9ea090e167ca6 Modification du texte
commit 56eb430b9f47772226081a30efc3b70f5825212 Affichage du texte
commit 5a0fb3ca400f10b57cda28b2e64f32417d29276 Ajout du fichier main
```

En passant `oneline` à l'option `--pretty` de la commande `git log`, le résultat obtenu affiche chaque commit de l'historique sur une seule ligne. Chaque ligne contient alors l'identifiant unique du commit, la branche sur laquelle il se trouve, et son commentaire.

```
1 git log --pretty=short
```

```
$ git log --pretty=short
commit 58d17bc325b5d798d2029f17a4e3d28fbd5d5653 (HEAD -> master)
Author: JohnDoeTest <JohnDoeTest@test.com>

    Nouvel affichage

commit 7a9e9b26351335e5663f7975d4c9ea090e167ca6
Author: JohnDoeTest <JohnDoeTest@test.com>

    Modification du texte

commit 56eb430b9f47772226081a30efc3b70f5825212
Author: JohnDoeTest <JohnDoeTest@test.com>

    Ajout du fichier index

commit 5a0fb3ca400f10b57cda28b2e64f32417d29276
Author: JohnDoeTest <JohnDoeTest@test.com>

    Affichage du texte

commit 5a0fb3ca400f10b57cda28b2e64f32417d29276
Author: JohnDoeTest <JohnDoeTest@test.com>

    Ajout du fichier main
```

La commande `git log --pretty=short` affiche sur plusieurs lignes, pour chaque commit, son identifiant unique avec la branche concernée, le nom de l'auteur du commit et le commentaire associé.

```
1 git log --pretty=fuller
```

```
$ git log --pretty=fuller
commit 547164a (HEAD -> master)
Author: JohnDoeTest <JohnDoeTest@test.com>
AuthorDate: Fri Apr 3 10:24:15 2020 +0200
Commit: JohnDoeTest <JohnDoeTest@test.com>
CommitDate: Fri Apr 3 11:23:45 2020 +0200

Nouvel affichage

commit 50d178c
Author: JohnDoeTest <JohnDoeTest@test.com>
AuthorDate: Fri Apr 3 10:23:22 2020 +0200
Commit: JohnDoeTest <JohnDoeTest@test.com>
CommitDate: Fri Apr 3 11:23:45 2020 +0200

Modification du texte

commit 7a9e9b2
Author: JohnDoeTest <JohnDoeTest@test.com>
AuthorDate: Fri Apr 3 11:04:40 2020 +0200
Commit: JohnDoeTest <JohnDoeTest@test.com>
CommitDate: Fri Apr 3 11:23:24 2020 +0200

Ajout du fichier index

commit 56eb436
Author: JohnDoeTest <JohnDoeTest@test.com>
AuthorDate: Fri Apr 3 10:05:09 2020 +0200
Commit: JohnDoeTest <JohnDoeTest@test.com>
CommitDate: Fri Apr 3 11:23:24 2020 +0200

Affichage du texte

commit 5a0fb3c
Author: JohnDoeTest <JohnDoeTest@test.com>
AuthorDate: Fri Apr 3 10:02:24 2020 +0200
Commit: JohnDoeTest <JohnDoeTest@test.com>
CommitDate: Fri Apr 3 11:23:14 2020 +0200

Ajout du fichier main
```

L'utilisation de `git log --pretty` avec l'option `fuller` va afficher des informations supplémentaires à celles déjà présentes avec l'option `short`.

Remarque

Il existe un raccourci à l'option `--pretty=oneline`. En effet, `git log` accepte directement l'option `--oneline`.

Celle-ci cumule l'option `--pretty=oneline` et l'option `--abbrev-commit`, ce qui permet de n'afficher que les premiers caractères de l'identifiant unique du commit.

Exemple

```
1 git log --oneline
```

```
$ git log --oneline
547164a (HEAD -> master) Nouvel affichage
50d178c Modification du texte
7a9e9b2 Ajout du fichier index
56eb436 Affichage du texte
5a0fb3c Ajout du fichier main
```

La commande `git log --oneline` affiche les résultats comme avec l'utilisation de `git log --pretty=oneline`, en tronquant l'identifiant des commits.

Méthode Formater l'affichage de l'historique

En plus de ces formatages préconfigurés, l'option `--pretty` peut prendre en paramètre l'option `format`, qui permet de définir un format de sortie personnalisé adapté aux besoins.

La définition d'un format se fait grâce à une chaîne de caractères contenant les différents masques des informations à afficher.

La liste des masques disponibles peut être trouvée ici : https://git-scm.com/docs/git-log#_pretty_formats.

Exemple

```
1 git log --pretty=format:"%h - %an, %ad : %s"
```

```
$ git log --pretty=format:"%h - %an, %ad : %s"
547164a - JohnDoeTest, Fri Apr 3 10:24:15 2020 +0200 : Nouvel affichage
50d178c - JohnDoeTest, Fri Apr 3 10:23:22 2020 +0200 : Modification du texte
7a9e9b2 - JohnDoeTest, Fri Apr 3 11:04:40 2020 +0200 : Ajout du fichier index
56eb436 - JohnDoeTest, Fri Apr 3 10:05:09 2020 +0200 : Affichage du texte
5a0fb3c - JohnDoeTest, Fri Apr 3 10:02:24 2020 +0200 : Ajout du fichier main
```

Les masques utilisés dans la commande `git log --pretty=format:"%h - %an, %ad : %s"` affichent un résumé de l'identifiant du commit suivi du nom de l'auteur, de la date au format date définie dans la configuration de Git, puis du commentaire du commit.


```
1 git log --pretty=format:"%h : %s"
```

```
$ git log --pretty=format:"%h : %s"
547164a : Nouvel affichage
58d170c : Modification du texte
7a9e9b2 : Ajout du fichier index
56eb436 : Affichage du texte
5a0fb3c : Ajout du fichier main
```

Le formatage utilisé avec `git log --pretty=format:"%h : %s"` affiche une version courte des identifiants des commits et les commentaires associés.

Fondamental À retenir

- L'option `--pretty` de la commande `git log` permet d'avoir le contrôle sur les résultats affichés par la commande.
- Il est possible de lui indiquer des formats prédéfinis, tels que `oneline` ou `fuller`, ou en spécifiant le format souhaité avec l'option `format`.

V. Rechercher, comprendre et filtrer l'historique

Objectifs

- Apprendre à visualiser clairement les imbrications de branches dans l'historique de Git
- Apprendre à limiter les résultats obtenus en fonctions de critères

Mise en situation

Il est courant, au sein d'un projet versionné avec Git, d'utiliser le système de branches qu'il offre pour segmenter les différents développements.

De plus, lorsqu'un projet de développement prend de l'ampleur, les éléments contenus dans l'historique Git deviennent de plus en plus importants, ce qui complique encore la recherche d'éléments précis dans l'historique.

Nous allons voir ici comment obtenir un affichage représentant graphiquement l'enchaînement des branches de l'historique, en utilisant la commande `git log`.

Nous présenterons également quelques moyens de réaliser des recherches directement dans l'historique.

Méthode Représenter graphiquement l'historique des modifications

Pour faciliter la compréhension de l'imbrication des différentes branches et des commits qu'elles contiennent au sein de l'historique d'un projet, la commande `git log` propose l'option `--graph`.

Celle-ci va afficher l'historique sous la forme d'un graphique représentant les branches existantes et les commits sur chacune d'entre elles.

Exemple

```
1 git log --graph --pretty=format:"%h : %s"
```

```
$ git log --graph --pretty=format:"%h : %s"
* 08198ae : Merge branch 'exemple_fetch_branch'
* aff9cc9 : commit 2
* 6a01fd8 : Create exemple.php
* bbc1496 : file add
* 22d2279 : Create exemplePull.php
* 2a28d1a : Initial commit
```

La commande `git log --graph --pretty=format:"%h : %s"` affiche l'arborescence des branches et, pour chacun des commits, son identifiant au format court et le commentaire qui lui est associé.

Méthode

Rechercher dans l'historique

La commande `git log` offre la possibilité de réduire les résultats affichés en fonction de paramètres limitants.

Il existe par exemple l'option `--since` pour rechercher des commits en fonction de leur date, `--author` pour effectuer une recherche selon l'auteur des commits, ou `-S` pour n'afficher que les commits dont les modifications contiennent une chaîne de caractères donnée.

Exemple

```
1 git log --since=yesterday --pretty=format:"%h : %s"
```

```
$ git log --since=yesterday --pretty=format:"%h : %s"
08198ae : Merge branch 'exemple_fetch_branch'
```

La commande `git log --since=yesterday --pretty=format:"%h : %s"` affiche tous les commits depuis la veille, et les formate pour n'afficher que leur identifiant au format court et leur commentaire.

```
1 git log --since=2.days --pretty=format:"%h : %s"
```

```
$ git log --since=2.days --pretty=format:"%h : %s"
08198ae : Merge branch 'exemple_fetch_branch'
bbc1496 : file add
22d2279 : Create exemplePull.php
aff9cc9 : commit 2
6a01fd8 : Create exemple.php
2a28d1a : Initial commit
```

La commande `git log --since=2.days --pretty=format:"%h : %s"` affiche les commits dans le même format que précédemment, mais affiche tous les commits jusqu'à deux jours en arrière.

```
1 git log -SHello -p
```

```
$ git log -SHello -p
commit 56eb436b9f47772226001a30efc3b70f5825212
Author: JohnDoeTest <JohnDoeTest@test.com>
Date: Fri Apr 3 10:05:09 2020 +0200

Affichage du texte

diff --git a/main.php b/main.php
index e69de29..9749df5 100644
--- a/main.php
+++ b/main.php
@@ -0,0 +1,2 @@
+<?php
+echo "Hello world !";
\ No newline at end of file
```

La commande `git log -SHello -p` affiche les commits dont les modifications contiennent la chaîne de caractères *Hello* et affiche les informations par défaut des commits, puis montre le détail des modifications qu'ils contiennent.

Fondamental

À retenir

- La commande `git log --graph` permet d'afficher sous la forme d'un graphique les arborescences des branches contenues dans l'historique.
- Il est possible d'effectuer des recherches limitant le nombre de commits affichés avec certaines options.
- On retrouve des options telles que `--since` pour les recherches sur la date, `--author` pour filtrer un auteur, ou encore `-S` pour rechercher les commits dont les modifications contiennent une chaîne de caractères à préciser.

VI. Exercice : Appliquez la notion

Question

[solution n°2 p.15]

Dans un nouveau répertoire, clonez le projet suivant : <https://github.com/laravel/laravel.git>

Définissez une commande qui permettra d'afficher l'historique de tous les commits depuis 30 jours, ainsi que les branches dont ils proviennent.

Ces messages seront affichés sous la forme : `identifiant_court - message_commit - auteur, le date`.

VII. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°3 p.15]

Exercice

Grâce à quelle commande est-il possible de visualiser l'historique des commits présents sur la branche courante ?

- ☐ git history
- ☐ git commit
- ☐ git log

Exercice

Grâce à quel paramètre est-il possible d'afficher le détail des modifications dans l'historique ?

- ☐ -p
- ☐ -n
- ☐ --with-details

Exercice

Que permettra d'exécuter la commande suivante : `git log -3` ?

- ☐ Supprimer les 3 derniers commits du dépôt
- ☐ Afficher les 3 derniers commits du dépôt
- ☐ Afficher les 3 premiers commits du dépôt

Exercice

La commande `git log --stat` permet...

- ☐ D'afficher les statistiques globales du dépôt
- ☐ D'afficher, pour chaque commit, un résumé concis des modifications

Exercice

Grâce à quelle option est-il possible de formater l'affichage de l'historique ?

Exercice

Quel sera l'affichage de la commande suivante : `git log --pretty=format:"%h : %s" ?`

- ☐ Identifiant version courte : Message de commit
- ☐ Identifiant version courte : Auteur
- ☐ Identifiant version courte : Date de commit

Exercice

Laquelle de ces options permet d'afficher le plus de détails ?

- ☐ `--pretty=oneline`
- ☐ `--pretty=short`
- ☐ `--pretty=fuller`

Exercice

Grâce à quelle option est-il possible de représenter l'imbrication des branches dans l'historique ?

Exercice

Parmi ces options, lesquelles est-il possible d'utiliser pour filtrer l'affichage ?

- ☐ `--pretty=`
- ☐ `--author=`
- ☐ `--graph=`
- ☐ `--since`
- ☐ `-S`

Exercice

Pour ajouter un alias Git, quelle syntaxe est-il nécessaire de respecter ?

- ☐ `git config --global alias.{nom_alias} {commande_git}`
- ☐ `git config --global {commande_git} alias.{nom_alias}`
- ☐ `git add-alias --global alias.{nom_alias} {commande_git}`

B. Exercice : Défi

À ce stade, les membres de votre équipe vous considèrent comme leur référent concernant l'utilisation de Git.

Votre expertise étant reconnue de tous, vous êtes chargé de mettre à disposition une documentation succincte permettant au reste de l'équipe de se servir au mieux de leur historique Git.

Question

[solution n°4 p.17]

Vous allez devoir créer un alias permettant à votre équipe de disposer d'un historique dont le format sera le même pour tout le monde, à savoir :

`identifiant_court_commit` [de couleur rouge] : message de commit (il y a X [de couleur verte]) - auteur [de couleur bleue]

L'historique des branches doit également être affiché.

Vous devrez également fournir quelques exemples de commandes permettant d'effectuer les recherches suivantes :

- Afficher les commits présents sur une branche et pas une autre
- Filtrer les commits depuis une période donnée : 1 semaine, 2 mois
- Filtrer les commits par auteur
- Filtrer les commits selon leur contenu

Vous pouvez effectuer vos tests dans l'un des dépôts créés précédemment, par exemple dans le projet *Symfony*.

Solutions des exercices

p. 7 Solution n°1

```
1 # Définition de l'alias
2 git config --global alias.last-5 'log --stat -5'
```

p. 11 Solution n°2

```
1 git clone https://github.com/laravel/laravel.git && cd laravel
2 git checkout 8.x
3 git log --graph --since=30.days --pretty=format:"%h - %s - %an - %ad"
```

```
$ git log --graph --since=30.days --pretty=format:"%h - %s - %an - %ad"
* eb080e5c - Merge branch 'master' into develop - Dries Vints - Mon Apr 27 16:33:05 2020 +0200
* 785405b3 - Merge branch '6.x' - Dries Vints - Mon Apr 27 16:32:54 2020 +0200
* d7d342c2 - Disable webpack-dev-server host check (#5288) - Nabil Muh. Firdaus - Mon Apr 27 20:38:20 2020 +0700
* fac16a7e - Update DatabaseSeeder.php - Taylor Otwell - Tue Apr 21 14:47:10 2020 -0500
* 2149e8c2 - update seeders - Taylor Otwell - Tue Apr 21 14:44:37 2020 -0500
* 5f6f5c92 - add model - Taylor Otwell - Tue Apr 21 12:41:27 2020 -0500
* b9af2b22 - Use new factories - Taylor Otwell - Tue Apr 21 10:12:39 2020 -0500
* 0e01834b - add sponsor link - Taylor Otwell - Fri Apr 17 10:36:39 2020 -0500
* 919a0491 - Merge branch 'master' into develop - Dries Vints - Thu Apr 16 09:47:44 2020 +0200
* 3cdf0ce - Merge branch '6.x' - Dries Vints - Thu Apr 16 09:47:28 2020 +0200
* 8aab222c - Update CHANGELOG.md - Dries Vints - Thu Apr 16 09:45:26 2020 +0200
* 8be1fd40 - Merge branch 'master' into develop - Dries Vints - Wed Apr 15 13:47:54 2020 +0200
* b5f008a8 - Update CHANGELOG.md - Dries Vints - Wed Apr 15 13:46:52 2020 +0200
* 103fc0b4 - Merge pull request #5278 from voyula/patch-1 - Dries Vints - Tue Apr 14 10:58:32 2020 +0200
* 0d23e5f7 - [7.x] Normalize Style - Musa - Tue Apr 14 11:56:29 2020 +0300
* d53c56e4 - remove unneeded parent boot calls - Taylor Otwell - Tue Apr 14 16:14:14 2020 -0500
```

Exercice p. 11 Solution n°3

Exercice

Grâce à quelle commande est-il possible de visualiser l'historique des commits présents sur la branche courante ?

- ☐ git history
- ☐ git commit
- ☒ git log

Exercice

Grâce à quel paramètre est-il possible d'afficher le détail des modifications dans l'historique ?

- ☒ -p
 - ☐ -n
 - ☐ --with-details
- Ce paramètre n'existe pas.*

Exercice

Que permettra d'exécuter la commande suivante : `git log -3` ?

- ☐ Supprimer les 3 derniers commits du dépôt
- ☒ Afficher les 3 derniers commits du dépôt
- ☐ Afficher les 3 premiers commits du dépôt

Exercice

La commande `git log --stat` permet...

- ☐ D'afficher les statistiques globales du dépôt
- ☒ D'afficher, pour chaque commit, un résumé concis des modifications

Exercice

Grâce à quelle option est-il possible de formater l'affichage de l'historique ?

`pretty`

Exercice

Quel sera l'affichage de la commande suivante : `git log --pretty=format:"%h : %s" ?`

- ☒ Identifiant version courte : Message de commit
- ☐ Identifiant version courte : Auteur
- ☐ Identifiant version courte : Date de commit

Exercice

Laquelle de ces options permet d'afficher le plus de détails ?

- ☐ `--pretty=oneline`
- ☐ `--pretty=short`
- ☒ `--pretty=fuller`

Exercice

Grâce à quelle option est-il possible de représenter l'imbrication des branches dans l'historique ?

`graph`

Exercice

Parmi ces options, lesquelles est-il possible d'utiliser pour filtrer l'affichage ?

- ☐ `--pretty=`
- ☒ `--author=`
- ☐ `--graph=`
- ☒ `--since`
- ☒ `-S`

Exercice

Pour ajouter un alias Git, quelle syntaxe est-il nécessaire de respecter ?

- ☒ `git config --global alias.{nom_alias} {commande_git}`
- ☐ `git config --global {commande_git} alias.{nom_alias}`
- ☐ `git add-alias --global alias.{nom_alias} {commande_git}`

p. 12 Solution n°4

```
1 #Commande permettant l'affichage tel que demandé
2 git log --graph --pretty=format:'%Cred%h%Creset -%d %s (%Cgreen%cr%Creset) - %Cblue%an' --
  abbrev-commit
3
4 # Ajout en tant qu'alias, à nommer selon votre convenance, ici log-defi
5 git config --global alias.log-defi "log --graph --pretty=format:'%Cred%h%Creset -%d %s
  (%Cgreen%cr%Creset) - %Cblue%an' --abbrev-commit"
6
7 # Afficher les commits présents sur une branche et pas une autre
8 git log-defi branche1..branche2
9
10 # Afficher les commits de la dernière semaine
11 git log-defi --since="1 week ago"
12
13 # Afficher les commits des deux derniers mois
14 git log-defi --since="2 months ago"
15
16 # Afficher les commits d'un auteur spécifique
17 git log-defi --author="Fabien"
18
19 # Afficher les commits contenant un message spécifique
20 git log-defi --grep="contenu à rechercher"
21
22
```