Git Stash

Table of Contents

git stash stocke (ou stashe) temporairement les changements apportés à une copie de travail pour que qu'on puisse effectuer d'autres tâches, revenir et les réappliquer par la suite. Le **stash** est utile si on a besoin de changer rapidement de contexte et de travailler sur autre chose, mais qu'on est en plein dans un changement de code et qu'on n'est pas tout à fait prêt à commiter.

Fonctionnement du stash Git

Les **stash** sont en fait encodés dans le dépôt en tant qu'objets de commit. La référence spéciale à `.git/refs/stash` pointe vers le **stash** le plus récent, et les stashes créés au préalable sont référencés par le `reflog` de **stash**. C'est pour cela que lorsqu'on faite référence à des **stash** grâce à stash@{n}:, on fait en fait référence à l'entrée **n** du reflog pour la référence **stash**. Étant donné qu'un stash n'est qu'un commit, on peut pouvez l'inspecter grâce à la commande `git log` :

```
$ git log --oneline --graph stash@{0}
*-. 953ddde WIP on master: 5002d47 our new homepage
|\ \
| | * 24b35a1 untracked files on master: 5002d47 our new homepage
| * 7023dd4 index on master: 5002d47 our new homepage
|/
* 5002d47 our new homepage
```

En fonction de l'élément stashé, une opération git stash unique crée deux ou trois commits. Les commits dans le diagramme ci-dessus représentent :

- **stash@{0}**, un nouveau commit permettant de stocker les fichiers suivis qui se trouvaient dans la copie de travail lorsqu'on a exécuté `git stash` ;
- **le premier parent de stash@{0}**, le commit préexistant qui se trouvait au niveau de HEAD lorsqu'on a exécuté `git stash` ;

- **le deuxième parent de stash@{0}**, un nouveau commit représentant l'index au moment où on a exécuté `git stash` ;
- **le troisième parent de stash@{0}**, un nouveau commit représentant les fichiers non suivis qui se trouvaient dans la copie de travail lorsqu'on a exécuté `git stash`. Ce troisième parent est uniquement créé si :
    - la copie de travail contenait véritablement des fichiers non suivis ; et
    - on a spécifié l'option `--include-untracked` ou `--al` lorsqu'on a appelé `git stash`.

Voici comment git stash encode l'arborescence de travail et l'index sous forme de commits :

- Avant un stash, l'arborescence de travail peut contenir des changements apportés aux fichiers suivis, non suivis et ignorés. Certains de ces changements peuvent également être stagés dans l'index.
- Lorsqu'on appelle `git stash`, on encode tous les changements apportés aux fichiers suivis sous forme de deux nouveaux commits: un pour les changements non stagés et un pour les changements stagés dans l'index. La réf `refs/stash` spéciale est mise à jour de sorte à pointer vers eux.
- L'option `--include-untracked` encode également tout changement apporté aux fichiers non suivis sous forme de commit supplémentaire.
- L'option `--all` inclut les changements apportés à tous les fichiers ignorés et aux fichiers non suivis dans le même commit.

Lorsqu'on exécute `git stash pop`, les changements introduits par les commits ci-dessus sont utilisés pour mettre à jour la copie de travail et l'index, et le reflog du stash est remanié de sorte à supprimer le commit apparu. Les commits apparus ne sont pas immédiatement supprimés, mais deviennent candidats pour une prochaine opération « garbage collection ».

Utilisation du stash

Faire un stash d'un travail

La commande `git stash` prend less changements non commités (stagés et non stagés), les enregistre pour une utilisation ultérieure, puis les annule dans la copie de travail. Par exemple :

```
$ git status
On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
$ git stash
Saved working directory and index state WIP on master: 5002d47 our new
homepage
HEAD is now at 5002d47 our new homepage
$ git status
On branch master
nothing to commit, working tree clean
```

À ce stade, on peut procéder à des changements, créer des commits, basculer entre des branches et effectuer toute autre opération Git, puis revenir et réappliquer le stash lorsqu'on est prêt.

Le stash est local pour le dépôt Git. Les stash ne sont pas transférés au serveur lors d'un push.

Appliquer les changements stashés

On peut réappliquer les changements stashés au préalable grâce à la commande `git stash pop` :

```
$ git status
On branch master
nothing to commit, working tree clean
$ git stash pop
On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
Dropped refs/stash@{0} (32b3aa1d185dfe6d57b3c3cc3b32cbf3e380cc6a)
```

L'éclatement du stash supprime les changements qu'il contient et les réapplique à la copie de travail.

On peut également réappliquer les changements à la copie de travail et les conserver dans le stash grâce à la commande `git stash apply`:

```
$ git stash apply
On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
```

C'est utile si on souhaite appliquer les mêmes changements stashés à plusieurs branches.

Ppar défaut, Git ne fera pas de stash des changements apportés aux fichiers non suivis ou ignorés.

Faire un stash des fichiers non trackés ou ignorés

Par défaut, l'exécution de git stash stashera :

- les changements ajoutés à votre index (stagés) ;
- les changements apportés aux fichiers actuellement suivis par Git (non stagés).

Mais cette commande ne stashera pas :

- les nouveaux fichiers dans votre copie de travail qui n'ont pas encore été stagés ;
- les fichiers qui ont été ignorés.

Par conséquent, si on ajoute un troisième fichier à l'exemple ci-dessus, mais qu'on ne le stage pas (autrement dit, on n'exécute pas git add), git stash ne les stashe pas.

```
$ script.js
$ git status
On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
Untracked files:
script.js
$ git stash
Saved working directory and index state WIP on master: 5002d47 our new
homepage
HEAD is now at 5002d47 our new homepage
$ git status
On branch master
Untracked files:
script.js
```

Si on ajoute l'option `-u` (ou `--include-untracked`), on demande à git stash de stasher également les fichiers non suivis :

```
$ git status
On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
Untracked files:
script.js
$ git stash -u
Saved working directory and index state WIP on master: 5002d47 our new
homepage
HEAD is now at 5002d47 our new homepage
$ git status
On branch master
nothing to commit, working tree clean
```

On peut également inclure des changements aux fichiers ignorés en transmettant l'option `-a` (ou `--all`) lorsqu'on exécute `git stash`.

Gérer plusieurs stashes

On n'est pas limité à un stash unique. On peut exécuter `git stash` à plusieurs reprises pour créer différents stash, puis utiliser `git stash list` pour les consulter. Par défaut, les stash sont simplement identifiés comme « WIP » (Work in progress, travail en cours) en haut de la branche et du commit à partir duquel on a créé le stash. Après un certain temps, il peut être difficile de se souvenir de ce que contient chaque stash :

```
$ git stash list
stash@{0}: WIP on master: 5002d47 our new homepage
stash@{1}: WIP on master: 5002d47 our new homepage
stash@{2}: WIP on master: 5002d47 our new homepage
```

Pour donner plus de contexte, il peut être judicieux d'annoter vos stash avec une description grâce à git stash save “message” :

```
$ git stash save "add style to our site"
Saved working directory and index state On master: add style to our site
HEAD is now at 5002d47 our new homepage
$ git stash list
stash@{0}: On master: add style to our site
stash@{1}: WIP on master: 5002d47 our new homepage
stash@{2}: WIP on master: 5002d47 our new homepage
```

Par défaut, git stash pop réappliquera le stash créé le plus récemment : `stash@{0}`

On peut choisir le stash à réappliquer en transmettant son identifiant en tant que dernier argument, par exemple :

```
$ git stash pop stash@{2}
```

Affichage des comparaisons entre stashes

On peut afficher un résumé d'un stash grâce à la commande git stash show :

```
$ git stash show
index.html | 1 +
style.css | 3 +++
2 files changed, 4 insertions(+)
```

Pour afficher la comparaison complète d'un stashinon, ajouter l'option `-p` (ou `--patch`):

```
$ git stash show -p
diff --git a/style.css b/style.css
new file mode 100644
```

```
index 0000000..d92368b
--- /dev/null
+++ b/style.css
@@ -0,0 +1,3 @@
+* {
+ text-decoration: blink;
+}
diff --git a/index.html b/index.html
index 9daeafb..ebdcbd2 100644
--- a/index.html
+++ b/index.html
@@ -1 +1,2 @@
+<link rel="stylesheet" href="style.css"/>
```

Stashes partiels

On peut également choisir de stasher uniquement un fichier, une collection de fichiers ou des changements individuels depuis les fichiers. Si on indique l'option `-p` (ou `--patch`) à git stash, elle itérera chaque « bloc » modifié dans la copie de travail et demandera si on souhaite le stasher :

```
$ git stash -p
diff --git a/style.css b/style.css
new file mode 100644
index 0000000..d92368b
--- /dev/null
+++ b/style.css
@@ -0,0 +1,3 @@
+* {
+ text-decoration: blink;
+}
Stash this hunk [y,n,q,a,d,/,e,?]? y
diff --git a/index.html b/index.html
index 9daeafb..ebdcbd2 100644
--- a/index.html
+++ b/index.html
@@ -1 +1,2 @@
+<link rel="stylesheet" href="style.css"/>
Stash this hunk [y,n,q,a,d,/,e,?]? n
```

Les commandes de bloc, les plus fréquemment utilisées sont :

Commande	Description
/	Rechercher un bloc par regex
?	Aide
n	Ne pas stasher ce bloc
q	Quitter (tous les blocs déjà sélectionnés seront stashés)
s	Diviser ce bloc en plus petits
y	Stasher ce bloc

Il n'existe aucune commande « d'annulation », mais appuyer sur `CTRL-C(SIGINT)` annulera le processus de stash.

Créer une branche depuis un stash

Si les changements sur la branche divergent de ceux dans le stash, on risque de rencontrer des conflits lors de l'éclatement ou de l'application du stash. À la place, on peut utiliser `git stash branch` pour créer une branche à laquelle appliquer vos changements stashés :

```
$ git stash branch add-stylesheet stash@{1}
Switched to a new branch 'add-stylesheet'
On branch add-stylesheet
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
Dropped refs/stash@{1} (32b3aa1d185dfe6d57b3c3cc3b32cbf3e380cc6a)
```

Une nouvelle branche basée sur le commit à partir duquel on a créé le stash fera l'objet d'un check-out, et les changements stashés y figureront.

Nettoyer un stash

Lorsqu'on n'a plus besoin d'un stash spécifique, on peut le supprimer grâce à la commande `git stash drop` :

```
$ git stash drop stash@{1}
Dropped stash@{1} (17e2697fd8251df6163117cb3d58c1f62a5e7cdb)
```

On peut également supprimer tous les stash grâce à la commande suivante :

```
$ git stash clear
```