Opérations et principes avancés de Git

Cet article de blog a été rédigé avec l'assistance de ChatGPT-4o. Le Keynote est disponible sur https://github.com/lzwjava/Keynotes.

Annuler

```
$ git commit --amend
$ git add *
$ git status
$ git reset HEAD CONTRIBUTING.md
Modifications non indexées après reset :
M CONTRIBUTING.md
$ git status
$ git checkout -- CONTRIBUTING.md
```

- Refaire un commit
- Annuler la mise en stage des fichiers
- Annuler les modifications apportées aux fichiers

Commande

- `git revert`, annule un commit en créant un nouveau commit qui inverse les modifications apportées par un commit précédent.
- `git cherry-pick`, sélectionne un ou plusieurs commits spécifiques d'une branche et les applique à une autre branche.
- `git rebase`, réapplique une série de commits sur une autre branche, permettant de réorganiser l'historique des commits.

filter-branch

- Supprimer un fichier de chaque commit
- Vous voulez le supprimer, mais il reste dans l'historique
- Comment supprimer un fichier de tous les commits ?

```
$ git filter-branch --tree-filter 'rm -f passwords.txt' HEAD
Réécriture de 6b9b3cf04e7c5686a9cb838c3f36a8cb6a0fc2bd (21/21)
La référence 'refs/heads/master' a été réécrite
```

Recherche

```
$ git grep -n gmtime_r
compat/gmtime.c:3:#undef gmtime_r
compat/gmtime.c:8:return git_gmtime_r(timep, &result);
compat/gmtime.c:11:struct tm *git_gmtime_r(const time_t *timep, struct tm *result)
compat/gmtime.c:16:ret = gmtime_r(timep, result);
compat/mingw.c:606:struct tm *gmtime_r(const time_t *timep, struct tm *result)
compat/mingw.h:162:struct tm *gmtime_r(const time_t *timep, struct tm *result);
date.c:429:if (gmtime_r(&now, &now_tm))
date.c:492:if (gmtime_r(&time, tm)) {
git-compat-util.h:721:struct tm *git_gmtime_r(const time_t *, struct tm *);
git-compat-util.h:723:#define gmtime_r git_gmtime_r
$ git log -SZLIB_BUF_MAX --oneline
e01503b zlib: allow feeding more than 4GB in one go
ef49a7a zlib: zlib can only process 4GB at a time
```

Répertoire .Git

- config, configuration du projet
- info, fichier .gitignore
- objects, tous les contenus de la base de données Git
- refs, pointeurs des branches
- HEAD, pointeur de la branche actuelle
- index, informations de la zone de staging (index)

Objets Git

```
$ git init test
Initialisé un dépôt Git vide dans /tmp/test/.git/
$ cd test
$ find .git/objects
.git/objects
```

```
.git/objects/info
.git/objects/pack
$ find .git/objects -type f

$ echo 'test content' | git hash-object -w --stdin
d670460b4b4aece5915caf5c68d12f560a9fe3e4
$ find .git/objects -type f
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
```

- `hash-object`, la commande pour sauvegarder les données dans le répertoire .git
- `-w`, écrire l'objet, sinon retourne seulement la clé
- `--stdin`, lire depuis l'entrée standard
- d670..., un checksum de 40 caractères
- `cat-file`, le couteau suisse pour visualiser les objets Git

```
$ git cat-file -p d670460b4b4aece5915caf5c68d12f560a9fe3e4
contenu de test

$ echo 'version 1' > test.txt
$ git hash-object -w test.txt
83baae61804e65cc73a7201a7252750c76066a30
$ echo 'version 2' > test.txt
$ git hash-object -w test.txt
1f7a7a472abf3dd9643fd615f6da379c4acb3e3a
$ find .git/objects -type f
.git/objects/1f/7a7a472abf3dd9643fd615f6da379c4acb3e3a
.git/objects/83/baae61804e65cc73a7201a7252750c76066a30
.git/objects/d6/70460b4b4aece5915caf5c68d12f560a9fe3e4
$ git cat-file -p 83baae61804e65cc73a7201a7252750c76066a30 > test.txt
$ cat test.txt
version 1
$ git cat-file -p 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a > test.txt
$ cat test.txt
version 2
$ git cat-file -t 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a
blob
$ git cat-file -p master^{tree}
```

```
100644 blob a906cb2a4a904a152e80877d4088654daad0c859 README
100644 blob 8f94139338f9404f26296befa88755fc2598c289 Rakefile
040000 tree 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0 lib
$ git cat-file -p 99f1a6d12cb4b6f19c8655fca46c3ecf317074e0
100644 blob 47c6340d6459e05787f644c2447d2595f5d3a54b simplegit.rb
Tree Objects

$ git update-index --add --cacheinfo 100644 \
83baae61804e65cc73a7201a7252750c76066a30 test.txt
update-index
```

- `update-index`, la commande pour créer un arbre
- `--add`, crée l'index
- `--cacheinfo`, lit depuis la base de données Git
- `100644`, mode du fichier, pour un fichier normal ; 100755 pour un fichier exécutable ; 120000 pour un lien symbolique
- 83baae, est le contenu précédent, version 1
- `\`, une commande sur une ligne divisée en deux lignes

write-tree

La commande `write-tree` est utilisée dans Git pour créer un arbre (tree) à partir de l'index actuel. Un arbre dans Git représente une structure de répertoires et de fichiers, similaire à un répertoire dans un système de fichiers. Cette commande est principalement utilisée dans des scripts ou des outils personnalisés pour manipuler directement les objets Git.

Utilisation de base

Pour créer un arbre à partir de l'index actuel, vous pouvez simplement exécuter :

```
git write-tree
```

Cette commande retourne l'identifiant SHA-1 de l'arbre créé. Cet identifiant peut ensuite être utilisé pour créer un commit ou pour d'autres opérations de bas niveau dans Git.

Exemple

Supposons que vous avez ajouté des fichiers à l'index avec `git add` et que vous souhaitez créer un arbre à partir de cet index :

```
git add .
git write-tree
```

Cela retournera un identifiant SHA-1, par exemple `4b825dc642cb6eb9a060e54bf8d69288fbee4904`. Cet identifiant représente l'arbre créé à partir de l'index actuel.

Utilisation avancée

La commande `write-tree` peut également être utilisée en combinaison avec d'autres commandes de bas niveau comme `commit-tree` pour créer des commits manuellement. Par exemple :

```
TREE=$(git write-tree)
COMMIT=$(echo "Initial commit" | git commit-tree $TREE)
git update-ref refs/heads/main $COMMIT
```

Dans cet exemple, nous créons d'abord un arbre à partir de l'index, puis nous créons un commit à partir de cet arbre, et enfin nous mettons à jour la référence de la branche `main` pour pointer vers ce nouveau commit.

Conclusion

La commande `write-tree` est un outil puissant pour ceux qui ont besoin de manipuler directement les objets Git. Bien qu'elle ne soit pas couramment utilisée dans les workflows Git quotidiens, elle est essentielle pour comprendre le fonctionnement interne de Git et pour créer des scripts ou des outils personnalisés.

```
$ git write-tree
d8329fc1cc938780ffdd9f94e0d364e0ea74f579
$ git cat-file -p d8329fc1cc938780ffdd9f94e0d364e0ea74f579
100644 blob 83baae61804e65cc73a7201a7252750c76066a30 test.txt
$ git cat-file -t d8329fc1cc938780ffdd9f94e0d364e0ea74f579
tree
```

- `write-tree`, écrit le contenu de la zone de staging dans un objet Tree.

read-tree

```
$ echo 'nouveau fichier' > new.txt
$ git update-index test.txt
$ git update-index --add new.txt
$ git write-tree
0155eb4229851634a0f03eb265b69f5a2d56f341
$ git cat-file -p 0155eb4229851634a0f03eb265b69f5a2d56f341
100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt
100644 blob 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a test.txt
$ git read-tree --prefix=bak d8329fc1cc938780ffdd9f94e0d364e0ea74f579
$ git write-tree
3c4e9cd789d88d8d89c1073707c3585e41b0e614
$ git cat-file -p 3c4e9cd789d88d8d89c1073707c3585e41b0e614
040000 tree d8329fc1cc938780ffdd9f94e0d364e0ea74f579 bak
100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt
100644 blob 1f7a7a472abf3dd9643fd615f6da379c4acb3e3a test.txt
```

Objets de Commit

```
$ echo 'premier commit' | git commit-tree d8329f
d5ffe1aa4b7b089eee03dccea5e0439ad6d72037
$ git cat-file -p d5ffe1aa4b7b089eee03dccea5e0439ad6d72037
tree d8329fc1cc938780ffdd9f94
```

e0d364e0ea74f579 author lzwjava lzwjava@gmail.com 1462090215 +0800 committer lzwjava lzwjava@gmail.com 1462090215 +0800 premier commit

```
- La valeur de hachage sera différente car la date de création et l'auteur sont différents.
```

$ echo ‘second commit’ | git commit-tree 0155eb -p d5ffe1aa4 e946d6367f07de45cac242dca7cd002f5eaa72b
$ echo ‘third commit’ | git commit-tree 3c4e9c -p e946d6 09490bf051c34b3693dfd5c7fb63dfe27b295904
$ git log –stat 09490 commit 09490bf051c34b3693dfd5c7fb63dfe27b295904 Author: lzwjava lzwjava@gmail.com Date: Sun May 1 16:38:55 2016 +0800 troisième commit bak/test.txt | 1 + 1 fichier modifié, 1 insertion(+) commit e946d6367f07de45cac242dca7cd002f5eaa72b1 Author: lzwjava lzwjava@gmail.com Date: Sun May 1 16:37:01 2016 +0800 second commit new.txt | 1 + test.txt | 2 +- 2 fichiers modifiés, 2 insertions(+), 1 suppression(-) commit d5ffe1aa4b7b089eee03dccea5e0439ad6d72037 Author: lzwjava lzwjava@gmail.com Date: Sun May 1 16:10:15 2016 +0800 premier commit test.txt | 1 + 1 fichier modifié, 1 insertion(+)

- `commit-tree`, le premier paramètre pointe vers un blob ou un arbre (tree)
- `-p` pointe vers le nœud parent

troisième commit deuxième commit premier commit arbre arbre arbre “version 2” “nouveau fichier” “version 1” 09490b e946d6 d5ffe1 d8329f 0155eb 3c4e9c 83baae fa49b0 1f7a7a test.txt new.txt test.txt new.txt test.txt bak

Le principe des branches

$ find .git/refs
.git/refs .git/refs/heads .git/refs/tags $ find .git/refs -type f
$ echo “09490bf051c34b3693dfd5c7fb63dfe27b295904” > .git/refs/heads/master $ git log –pretty=oneline master 09490bf051c34b3693dfd5c7fb63dfe27b295904 troisième commit e946d6367f07de45cac242dca7cd002f5eaa72b1 deuxième commit d5ffe1aa4b7b089eee03dccea5e0439ad6
premier commit $ git update-ref refs/heads/master 09490bf051c34b3693dfd5c7fb63dfe27b295904
$ git update-ref refs/heads/test e946d6367f07de45cac242dca7cd002f5eaa72b1 $ git branch
* master test

- Sauvegarder une référence de pointeur
- `update-ref`, utilisé pour modifier ou ajouter des références

troisième commit deuxième commit premier commit arbre arbre arbre “version 2” “nouveau fichier” “version 1” 09490b e946d6 d5ffe1 d8329f 0155eb 3c4e9c 83baae fa49b0 1f7a7a test.txt new.txt test.txt new.txt test.txt bak refs/heads/master refs/heads/test

$ cat .git/HEAD ref: refs/heads/master $ git symbolic-ref HEAD refs/heads/master $ git symbolic-ref HEAD refs/heads/test $ cat .git/HEAD ref: refs/heads/test

symbolic-ref

Principe des tags

$ git update-ref refs/tags/v1.0 e946d6367f07de45cac242dca7cd002f5eaa72b1 $ git tag v1.0 $ git tag -a v1.1 e946d6367f07de45cac242dca7cd002f5eaa72b1 -m ‘test tag’ $ cat .git/refs/tags/v1.1 2766532f03289bc5e158629a8b3faffa5f80b8b6 $ git cat-file -p 276653 object e946d6367f07de45cac242dca7cd002f5eaa72b1 type commit tag v1.1 tagger lzwjava lzwjava@gmail.com 1462103203 +0800 test tag

remotes

$ git remote add origin git@github.com:schacon/simplegit-progit.git $ git push origin master Décompte des objets: 11, fait. Compression des objets: 100% (5/5), fait. Écriture des objets: 100% (7/7), 716 octets, fait. Total 7 (delta 2), réutilisés 4 (delta 1) Vers git@github.com:schacon/simplegit-progit.git a11bef0..ca82a6d master -> master $ cat .git/refs/remotes/origin/master ca82a6dff817ec66f44342007202690a93763949

- Placé dans le répertoire `refs/remotes`
- La différence entre les références distantes (remotes) et les branches locales est que les références
- Vous pouvez utiliser `git checkout`, mais HEAD ne change pas, donc vous ne pouvez pas modifier les ré

$ curl https://raw.githubusercontent.com/mojombo/grit/master/lib/grit/repo.rb > repo.rb $ git checkout master $ git add repo.rb $ git commit -m 'ajout de repo.rb' [master 484a592] ajout de repo.rb 3 fichiers modifiés, 709 insertions(+), 2 suppressions(-) suppression du mode 100644 bak/test.txt création du mode 100644 repo.rb réécriture de test.txt (100%) $ git cat-file -p master^{tree} 100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt 100644 blob 033b4468fa6b2a9547a70d88d1bbe8bf3f9ed0d5 repo.rb 100644 blob e3f094f522629ae358806b17daf78246c27c007b test.txt $ git cat-file -s 033b4468fa6b2a9547a70d88d1bbe8 22044 $ echo '# testing' » repo.rb $ git commit -am 'modification de repo.rb un peu' [master 2431da6] modification de repo.rb un peu 1 fichier modifié, 1 insertion(+) $ git cat-file -p master^{tree} 100644 blob fa49b077972391ad58037050f2a75f74e3671e92 new.txt 100644 blob b042a60ef7dff760008df33cee372b945b6e884e repo.rb 100644 blob e3f094f522629ae358806b17daf78246c27c007b test.txt $ git cat-file -s b042a60ef7dff760008df33cee372b94 22054 $ git gc Dénombrement des objets: 18, fait. Compression delta utilisant jusqu'à 8 threads. Compression des objets: 100% (14/14), fait. Écriture des objets: 100% (18/18), fait. Total 18 (delta 3), réutilisés 0 (delta 0) $ find .git/objects -type f $ git verify-pack -v .git/objects/pack/pack-978e03944f5c581011e6998cd0e9e30000905586.idx

Refspec

$ git remote add origin https://github.com/schacon/simplegit-progit [remote "origin"] url = https://github.com/schacon/simplegit-progit fetch = +refs/heads/*:refs/remotes/origin/* $ git log origin/master $ git log remotes/origin/master $ git log refs/remotes/origin/master fetch = +refs/heads/master:refs/remotes/origin/master [remote "origin"] url = https://github.com/schacon/simplegit-progit fetch = +refs/heads/master:refs/remotes/origin/master fetch = +refs/heads/qa/*:refs/remotes/origin/qa/* push = refs/heads/master:refs/heads

`/qa/master`

- `<src>:<dst>`
- `refs/remotes` l'emplacement local

`https://github.com/schacon/simplegit-progit`

`https://github.com/schacon/simplegit-progit`

Références

- Pro Git